DVD GRATIS
KNOPPIX 6.3

LINUX SHELL
MANUAL PRÁCTICO

MANUAL PRÁCTICO

LINUX SHELL

MANUAL PRÁCTICO

EDICIÓN 2010 - 2011 6.95 €

300 DE LOS MEJORES COMANDOS BASH EXPLICADOS

LINUX Especial

Domina la Terminal

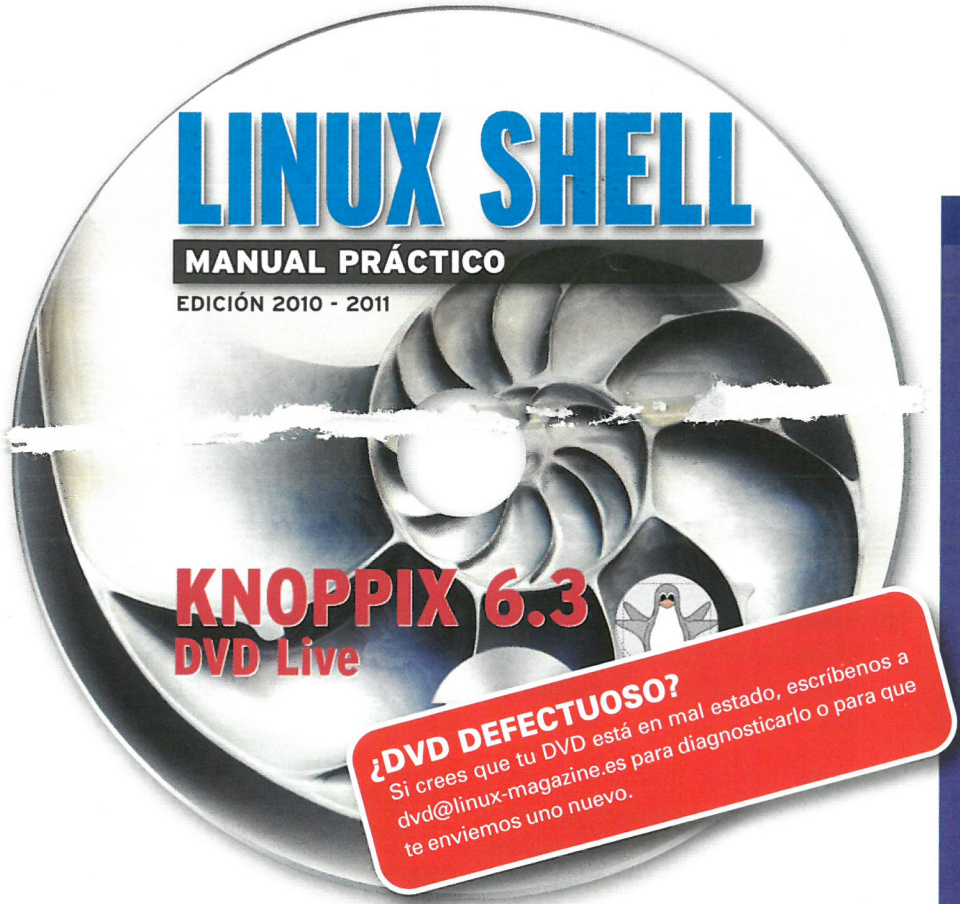
Descubre por qué los expertos en Linux prefieren la velocidad y la potencia de la línea de comandos

Tu Guía Esencial

Mantén está guía junto al ordenador como tu referencia para la línea de comandos

- Crea tus propios scripts Bash
- Configura particiones, permisos, dispositivos y cuentas de usuario
- Administra y monitoriza procesos

LINUX MAGAZINE
Ubuntu 10.10
Kubuntu 10.10
DVD de 2 caras
ubuntu 10.10 Maverick Meerkat
UBUNTU user
Número 01 - 6,95 €
EXPLORANDO EL MUNDO DE UBUNTU
MAVERICK MOLA
MEERKAT MODERNO
¿Que hay de nuevo en 10.10?
MÚSICA A TOPE
Damos una vuelta por l
tienda Ubuntu One
A TODA PASTIL
Los secretos del arran
rápido de Maverick
Launchpad al Descubie
OpenOffice Profesional
Crea un e-book
A LA VENTA OCTUBRE 2010
REVISTA UBUNTU USER 01
Incluye guía completa para descubrir Ubuntu y ayudar a nuevos usuarios a instalar, configurar y explorar su nuevo sistema Linux
Descúbrelo en www.ubuntu-user.es
¡DVD UBUNTU 10.10
MAVERICK MEERCAT GI
DESCUBRE M
Aprende con nuestros tutoriales sobre la terminal, seguridad y scripting
WWW.UBUNTU-USER.ES



LINUX DESDE LA SHELL

Antes de que llegaran los iconos, los menús y las ventanas que se mueven como gelatina en un entorno de escritorio Linux moderno, los usuarios se comunicaban con sus sistemas desde la línea de comandos. Con el paso de los años, la velocidad y eficiencia de este interfaz lo ha mantenido en boga en los círculos de expertos y profesionales. Muchos usuarios avanzados siguen prefiriendo el teclado al ratón y muchos otros dirán que explorar el entorno de la shell es la mejor manera de conocer de verdad Linux por dentro.

Este Manual Práctico de la Shell de Linux es el estudio más completo, accesible y amigable del Bourne Again Shell (Bash), el entorno de terminal más utilizado en sistemas Linux hoy en día. Aprenderás a navegar por la shell, a encontrar y manipular texto, a trabajar con expresiones regulares, a administrar hardware y usuarios y a personalizar tu entorno Bash. Aprenderás sobre las herramientas para todas estas tareas, a instalar y configurar nuevas herramientas, e incluso a crear las tuyas propias.

Mantén esta guía del Shell de Bash junto a tu ordenador como referencia permanente y completa del mundo en la ventana del terminal.

DVD Knoppix 6.3

Incluimos con este Manual Práctico de la Shell de Linux un DVD Live con el poderoso Knoppix 6.3. Knoppix ha sido durante años el compendio de herramientas preferido por los profesionales de la administración de sistemas, e incluye todo lo necesario (y bastantes cosas más) para seguir y aplicar los contenidos de este Manual Práctico. La distro Live de Knoppix además puede arrancarse y ejecutarse directamente desde el disco, por lo que no necesita instalar nada en su disco duro.

Para utilizar el DVD de Knoppix 6.3, inserte el disco en el dispositivo y rearranque su sistema. Si no aparece el menú del DVD, tiene que ajustar la BIOS para arrancar desde el disco. Para hacerlo, normalmente ha de pulsar una tecla durante las primeras fases del arranque del ordenador (habitualmente *F2* o *Supr*), buscar el menú que permite cambiar la secuencia de arranque de dispositivos y colocar el lector de DVDs por encima del disco duro. Posteriormente se guardan los cambios y se sale de la herramienta de configuración para que se reinicie el arranque. Como cada BIOS es diferente, recomendamos consultar la documentación del fabricante para realizar esta operación.

RECURSOS

[1] Entrada de la Wikipedia sobre Bash: *http://es.wikipedia.org/wiki/Bash*

[2] Sitio web de Bash: *http://www.gnu.org/software/bash/bash.html*

[3] Sitio web de Knoppix: *http://www.knoppix.net/*

BASH BÁSICO

6 A su Servicio
La línea de comandos suministra una ingente cantidad de información que no podemos obtener desde el escritorio.

10 Bien Organizado
Los comandos adecuados nos ayudan a administrar correctamente ficheros y directorios, algo que puede convertirse en un serio trabajo de reorganización y limpieza.

14 Objetos Perdidos
Un fichero perdido puede causar gran cantidad de trabajo y también puede plantear un riesgo de seguridad. Afortunadamente, Linux tiene algunas herramientas versátiles para encontrar estos "ficheros perdidos".

16 Cacería en el Disco Duro
¿Cómo encontramos exactamente un pasaje de un texto en nuestra base de datos digital? El comando de la shell grep nos puede ayudar a encontrar esa cita escurridiza.

18 Flujo de Datos
Tuberías, redireccionamientos y cadenas de comandos proporcionan a los usuarios un número casi infinito de opciones

20 Léelo Todo
Herramientas prácticas como cat, less, y head son ideales para visionar textos.

22 AWK
AWK permite realizar búsquedas y modificaciones en ficheros de texto formateados como tablas. Los administradores sienten una especial predilección por este lenguaje, que se encuentra instalado por defecto en la mayoría de los sistemas Unix.

25 Shell Personal
Consejos que nos ayudan a personalizar Bash.

CONFIGURACIÓN

28 Hardware Asistido
Aprenderemos algunos comandos que nos permitirán descubrir y configurar nuestro hardware.

32 Particiones Mágicas
Creamos y examinamos particiones de discos Linux con fdisk y parted.

35 Constructor
Mostramos cómo configurar y gestionar sistemas de ficheros con mkfs, df, du y fsck.

38 Atrapado en el Tiempo
Mantenemos el reloj de nuestro ordenador en marchar y sincronizamos la hora con un servidor de hora remoto.

CONFIGURACIÓN

41 Móntatelo
Examinamos las herramientas para montar y desmontar dispositivos de almacenamiento.

ADMINISTRACIÓN

44 ¡Acceso Permitido!
Vemos cómo crear y administrar cuentas de usuarios Linux desde la línea de comandos.

47 Abriendo la Puerta
Un sofisticado sistema de usuarios y permisos controla con precisión en Linux quién tiene acceso a qué. Podemos configurarlo todo en la línea de comandos con las herramientas chmod, chgrp y chown.

50 Identidad
su y sudo te permiten cambiar la identidad rápidamente desde la línea de comandos.

53 Vecinos
Mantener a los usuarios que comparten un mismo sistema en sus propios espacios e impedir que allanen los de los demás es una tarea para las herramientas de cuotas de disco.

56 Atareado
Te descubrimos todo acerca de los procesos de sistemas, cómo monitorizarlos, cómo controlarlos y las herramientas que te harán falta para ello. Y todo desde la línea de comandos.

OTRAS SECCIONES

03 DVD Knoppix 6.3
98 Información de Contacto

ADMINISTRACIÓN

60 A Prueba de Tontos
Los líos de las dependencias desaparecen con el sistema de paquetes Debian.

63 Paquetes Sabrosos
Yum, el administrador de paquetes de RPM, tiene sus propias ventajas sobre otras herramientas.

67 Imagen en el Espejo
Siempre que tengamos que hacer copias de seguridad o crear CDs de arranque del sistema operativo, dd y genisoimage nos ayudarán con todo el trabajo, y además lo harán con mucho estilo.

COMUNICACIÓN

70 A Través de la Red
Linux posee las herramientas adecuadas para buscar y encontrar en la red errores y abrir el camino a los paquetes de datos.

73 Zapador de Túneles
Siempre que necesitemos tunelizar conexiones para el envío de datos cifrados entre varios PCs, la herramienta SSH nos proporciona toda la funcionalidad requerida.

77 Remueve esos Datos
FTP y NcFTP son clientes FTP para la línea de comandos. Entre ambos poseen un amplio abanico de funcionalidades de trsnaferencia.

COMUNICACIÓN

80 Sincronízate
Rsync permite sincronizar datos tanto en un ordenador local como remoto. Incluso puede ser usado con SSH para el cifrado de información.

82 Unplugged
Un portátil, una WLAN abierta y las herramientas inalámbricas de la línea de comandos es todo cuanto se necesita para enviar un artículo desde cualquier lugar del mundo.

AUTOMATIZACIÓN

85 Tareas en Punto
Las utilidades cron y at ayudan a automatizar procesos en un sistema Linux.

88 Scripts a Medida
Con un par de trucos y algo de programación podremos ahorrar tiempo automatizando esas tareas que se repiten una y otra vez.

94 Programando Scripts
Con todas las novedades añadidas a las últimas versiones de Bash, rara vez hace falta el apoyo de programas externos.

96 Índice de Comandos

KNOPPIX 6,3

La mejor distro live del mundo

6.5 GBs de software Linux

Se ejecuta directamente desde el DVD

Incluye
- Google Chrome
- Adobe Acrobat Reader
- Adobe Flash
- Adrianne

y mucho más

Instalación sencilla en pendrives USB

Compatible con los repositorios de Debian

Ver pág. 3 para más detalles

Sasha Radosavljevic, 123RF

Descubrimos información del sistema desde la línea de comandos

A SU SERVICIO

La línea de comandos suministra una ingente cantidad de información que no podemos obtener desde el escritorio. **POR BRUCE BYFIELD**

La línea de comandos es una interfaz diseñada para confiarnos el control total del sistema cuando nos registramos como root. Como la información es necesaria para el control, no sorprende que la media de los escritorios libres contengan docenas de instrucciones diseñadas para decirnos exactamente qué está pasando en nuestro sistema. De hecho, muchos nos ofrecen tanta información que podemos pensar, "¡Se han pasado!"

Comenzaremos hablando de algunas de las instrucciones que pueden revelar la mayor parte de la información del sistema. Muchas carecen de una versión equivalente para el escritorio, pero si estamos intentando administrar un sistema, aunque sea a pequeña escala, deberíamos encontrarlas de utilidad.

Nótese que muchos de esos comandos no se encuentran disponibles en las cuentas de usuarios normales. Si probamos uno de ellos y el sistema afirma que no existe, deberemos intentar iniciar sesión como root o utilizar el comando con *sudo* para obtener los privilegios de administrador.

Información del Sistema General

El comando más básico para la información del sistema es *uname*, el cual nos da una gran cantidad de datos sobre el sistema (Figura 1). A pesar de que podemos elegir qué fragmentos de información muestre, habitualmente es más fácil listar simplemente todo lo que uname puede darnos introduciendo *uname -a*.

La salida de uname con la opción *-a* es una línea única con información separada por espacios. De izquierda a derecha encontramos el nombre general del kernel, el nombre de nuestra máquina, la versión del kernel y cuándo se compiló, y el sistema operativo que estamos usando.

Si deseamos conocer qué módulos del kernel se encuentran en uso, el comando a utilizar es *lsmod* - sin opciones (Figura 2). De hecho, es tan simple que incluso carece de página man. Este comando lista cada módulo del kernel, su tamaño y los módulos a los que hace referencia.

Memoria en Uso

En un sistema moderno, el espacio de disco duro y la memoria del sistema son asuntos menos preocupantes que lo que lo fueron hace cinco años. Aún así, las colecciones de canciones y películas pueden llenar incluso el disco duro más grande que existe, y con que ejecutemos un par de aplicaciones de diseño simultáneamente o nos pongamos a procesar sonido, pronto nos encontraremos en los límites de la RAM. Afortunadamente, disponemos de varios comandos que nos ayudan a hacerle un seguimiento al uso de memoria.

Para la memoria del disco duro, el comando básico es *df* (un resumen del espacio libre en disco), el cual en sí mismo lista el espacio en todas las particiones, aunque podemos especificar cuáles son las que deseamos que presente. Para perfeccionarlo más podemos usar la opción *-type=[sistemaficheros]* y presentar sólo aquellas particiones formateadas con un sistema de ficheros particular, como ext3 o ReiserFS. Otro parámetro útil es *-h* y *-H*, que los presenta en la forma denominada "legible por humanos". Por ejemplo, podría presentar un

```
bruce@nanday:/$ uname -a
Linux nanday 2.6.26-1-686-bigmem #1 SMP Sat Jan 10 19:13:22 UTC 2009 i686 GNU/Linux
```

Figura 1: El comando uname nos ofrece información sobre el sistema básico.

```
nanday:~# lsmod
Module                  Size  Used by
radeon                119488  0
drm                    66184  1 radeon
agpgart                29576  1 drm
battery                10948  0
ppdev                   7236  0
lp                      8932  0
ipv6                  235876  14
```

Figura 2: El comando lsmod lista cada módulo del kernel instalado.

```
bruce@nanday:/$ df -H --type=reiserfs
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       3.2G  378M  2.9G  12% /
/dev/sda9       276G   13G  264G   5% /home
/dev/sda8       3.6G   34M  3.6G   1% /tmp
/dev/sda5       105G  5.4G  100G   6% /usr
/dev/sda6       5.2G  948M  4.2G  19% /var
```

Figura 3: Para comprobar el espacio libre en disco duro usamos df.

tamaño de 269377692 como 270G (gigabytes).

Sea cual sea la opción que elijamos, df muestra la cantidad total de espacio en el disco duro, el espacio usado como cantidad y como porcentaje del total, el espacio libre y dónde se monta cada partición (Figura 3).

Para ver la memoria del sistema, la RAM combinada con el tamaño de la partición swap, usamos el comando *free* (Figura 4). Al igual que con el comando *df*, podemos especificar cómo se presentan las estadísticas de memoria, aunque, en el caso de la libre, podemos usar la opción *-m* para megabytes o *-g* para gigabytes (*-k* para kilobytes y *-b* para los bytes disponibles, también, aunque estas opciones no se utilizan mucho hoy en día). Además, con *-t* podemos añadir una línea a la salida que nos muestre los totales.

Si necesitamos seguir el uso de memoria seriamente, entonces *-s[SEGUNDOS]* actualiza los resultados en el intervalo dado por *[SEGUNDOS]*.

Información de Partición

La fuente de información más rápida para saber cómo se encuentran divididos nuestros discos duros es mostrar el contenido del fichero */etc/fstab*. Por ejemplo, en la máquina que uso, introduciendo el comando *less /etc/fstab* se obtienen los resultados que vemos en la Figura 5.

Veo rápidamente que tengo siete particiones de disco duro (incluyendo una para Windows que no está listada porque no la utilizo en */dev/sda1*, y excluyendo */dev/sda3*, que es una partición extendida y por consiguiente menos importante que las particiones que existen en él) y dónde se montan cada una de ellas, el sistema de ficheros empleado para montarlas y las opciones con las que son montadas. Para comprender los sistemas de ficheros, buscamos la entrada de la opción *-t* en la página man para el comando *mount*. La entrada de la página man también lista la opción *-o*, que nos da una explicación básica de las opciones de montaje.

Para saber específicamente qué está haciendo la partición swap en nuestro sistema, podemos utilizar el comando *swapon -s* para ver su tamaño y cuánto de ella se encuentra actualmente en uso (Figura 6).

Para información más detallada sobre las particiones, los usuarios más experimentados confían habitualmente en el comando *fdisk -l* (Figura 7). A pesar de que fdisk es el mismo comando básico para la creación y borrado de particiones, no hay de qué preocuparse: con el parámetro *-l*, sólo muestra dónde comienza y acaba cada partición y cómo están formateadas. El comando tiene la ventaja de listar */etc/fstab*, ya que muestra las particiones extendidas, además de las que no están montadas.

Sin embargo, incluso el venerable fdisk palidece cuando se le compara con GNU Parted, el editor de particiones que viene con la mayoría de las distribuciones actuales. Si introducimos el comando *parted*, podemos iniciar la shell propia de la aplicación (Figura 8), que tiene su propio juego de comandos básicos. Introduciendo *help* podemos obtener una lista completa de estos comandos, incluyendo *print*.

El resultado obtenido no es sólo información acerca de las características de nuestro disco duro, sino también de cada tipo de partición (por ejemplo, extendida o lógica) y de los indicadores que deberían estar en él, como cuál está marcado como autoarrancable. Cuando tenemos la información que deseamos, introducimos *quit* para abandonar la shell de Parted y volver a Bash.

```
bruce@nanday:/$ free -mt
free -mt
             total       used       free     shared    buffers     cached
Mem:          4052       1263       2789          0        201        497
-/+ buffers/cache:        564       3488
```

Figura 4: El comando free muestra la cantidad de memoria del sistema. La memoria del sistema es la cantidad de RAM más la partición swap.

```
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
/dev/sda2       /               reiserfs notail         0       1
/dev/sda9       /home           reiserfs defaults       0       2
/dev/sda8       /tmp            reiserfs defaults       0       2
/dev/sda5       /usr            reiserfs defaults       0       2
/dev/sda6       /var            reiserfs defaults       0       2
/dev/sda7       none            swap    sw              0       0
/dev/hda        /media/cdrom0   udf,iso9660 user,noauto     0       0
/dev/scd0       /media/cdrom1   udf,iso9660 user,noauto     0       0
/etc/fstab (END)
```

Figura 5: El fichero fstab posee información sobre los dispositivos montados actualmente.

```
nanday:~# swapon -s
Filename                        Type            Size    Used    Priority
/dev/sda7                       partition       2650684 0       -1
```

Figura 6: El comando swapon -s nos dice cómo está funcionando nuestra partición swap.

Más Información del Hardware del Sistema

Aparte de los discos duros y de los chips de los procesadores, el comando básico para los dispositivos de hardware es *lspci*. Como puede adivinarse por su nombre, lista los buses PCI que se encuentran instalados en el sistema. Si deseamos conocer qué tarjeta inalámbrica utiliza nuestro portátil o qué tarjeta de vídeo tenemos para poder encontrar sus correspondientes drivers o buscar información sobre ellas, entonces *lspci* es lo que buscamos.

```
nanday:~# fdisk -l

Disk /dev/sda: 500.1 GB, 500107862016 bytes
255 heads, 63 sectors/track, 60801 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x000607dc

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1       12748   102398278+   7  HPFS/NTFS
/dev/sda2           12749       13136     3116610   83  Linux
/dev/sda3           13137       60801   382869112+   5  Extended
/dev/sda5           13137       25882   102382213+  83  Linux
/dev/sda6           25883       26503     4988151   83  Linux
/dev/sda7           26504       26833     2650693+  82  Linux swap / Solaris
/dev/sda8           26834       27264     3461976   83  Linux
/dev/sda9           27265       60801   269385921   83  Linux
```

Figura 7: La manera tradicional de leer información de las particiones de nuestro disco duro es el comando fdisk -l.

Lo primero que necesitamos saber es que *lspci* muestra información en tres niveles de detalle o verbosidad. Para obtener el detalle mínimo introducimos solamente *lspci* (Figura 9). Para conseguir el nivel siguiente de detalle, *lspci -v*, y para el más detallado, *lspci -vv*.

```
(parted) print
Model: ATA ST3500630AS (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  Flags
 1      32.3kB  105GB   105GB   primary   ntfs         boot
 2      105GB   108GB   3191MB  primary   reiserfs
 3      108GB   500GB   392GB   extended
 5      108GB   213GB   105GB   logical   reiserfs
 6      213GB   218GB   5108MB  logical   reiserfs
 7      218GB   221GB   2714MB  logical   linux-swap
 8      221GB   224GB   3545MB  logical   reiserfs
 9      224GB   500GB   276GB   logical   reiserfs
```

Figura 8: El comando de impresión de la shell GNU Parted nos dirá cuanto queramos saber sobre las particiones de nuestro disco duro.

Lo siguiente que debemos saber es que, incluso al nivel de detalle más bajo, lspci produce más información que incluso lo que un terminal virtual de pantalla completa puede presentar de una sola vez. Por consiguiente, aconsejamos dirigir la salida a *less* o *more* de manera que podamos desplazarnos hacia arriba y hacia abajo en el texto cuando lo necesitemos: *lspci | less* o *lspci | more*. Si deseamos información acerca del dispositivo que estamos buscando, como su bus o slot, podemos filtrar la salida de lspci, aunque en muchos casos, ésta es exactamente la información que estamos intentado encontrar, así que este filtro es habitualmente de uso limitado.

El comando *lspci* también viene con un número de opciones útiles que proporcionan incluso más información. Por ejemplo, *-nn* muestra los nombres numéricos e ingleses de cada dispositivo, mientras que *-k*, los módulos del kernel asociados con el dispositivo. No importa qué opciones usemos, lspci nos ofrece tanta información, que solamente lo entenderían gurús del hardware más curtidos, pero si utilizamos los parámetros adecuados y nos tomamos nuestro tiempo, podremos encontrar la información que necesitamos.

Una Alternativa

Estos comandos no son los únicos que nos dan información sobre nuestro sistema – son de hecho los más básicos. Otros nos dan al menos parte de la misma información que los que hemos listado aquí; por ejemplo, podríamos usar *dnsdomainname* en lugar de *uname* para encontrar el nombre de la máquina. Sin embargo, los comandos que aquí hemos explicado son los más versátiles y deberían darnos la mayor parte de la información que deseamos o necesitamos.

No obstante, existe una alternativa lo bastante completa como para ser citada. Si necesitamos información que estos comandos no nos proporcionan, o si sentimos curiosidad por alguna alternativa, podemos intentar probar con */proc*, que es un pseudo sistema de ficheros generado durante el arranque que contiene información del kernel. Normalmente, el directorio *proc* es algo en lo que nunca es aconsejable toquetear, especialmente cuando nos registramos como root, porque un paso en falso podría hacer que nuestro sistema se colgara. Sin embargo, si nos limitamos a los comandos básicos como *cat* o *more*, no podremos dañarlo.

Si cambiamos a */proc* e introducimos el comando *ls*, encontraremos que el directorio consta de carpetas y ficheros. Si vemos los ficheros, podremos observar información variada acerca de nuestro sistema y cómo opera. Por ejemplo, *cat ./version* nos da información sobre la versión del kernel y la distribución que estamos usando, mientras que *cat ./cpuinfo* proporciona información detallada sobre el chipset de nuestro sistema. Incluso si no podemos adivinar qué tipo de información contiene cada fichero por su nombre, podemos optar a listar su contenido.

La ventaja de leer de */proc* es que tenemos pocos comandos que recordar. La desventaja, que la información no está organizada sistemáticamente para ser leída por humanos.

Sin embargo, entre los comandos habituales y */proc* deberíamos encontrar toda la información necesaria sobre nuestro software y hardware y mucho más. Ambas son maneras de usar la línea de comandos para obtener información que simplemente no está disponible desde el escritorio. ■

```
bruce@nanday:~$ lspci | more
00:00.0 RAM memory: nVidia Corporation C51 Host Bridge (rev a2)
00:00.1 RAM memory: nVidia Corporation C51 Memory Controller 0 (rev a2)
00:00.2 RAM memory: nVidia Corporation C51 Memory Controller 1 (rev a2)
00:00.3 RAM memory: nVidia Corporation C51 Memory Controller 5 (rev a2)
00:00.4 RAM memory: nVidia Corporation C51 Memory Controller 4 (rev a2)
00:00.5 RAM memory: nVidia Corporation C51 Host Bridge (rev a2)
00:00.6 RAM memory: nVidia Corporation C51 Memory Controller 3 (rev a2)
00:00.7 RAM memory: nVidia Corporation C51 Memory Controller 2 (rev a2)
00:02.0 PCI bridge: nVidia Corporation C51 PCI Express Bridge (rev a1)
00:03.0 PCI bridge: nVidia Corporation C51 PCI Express Bridge (rev a1)
00:04.0 PCI bridge: nVidia Corporation C51 PCI Express Bridge (rev a1)
```

Figura 9: El comando lspci describe el hardware de nuestro ordenador de manera exhaustiva.

Linux Magazine
ACADEMY

Soy root y lo puedo demostrar

Pruébalo gratis en academialinux.com

Formación en línea con...

- Preparación para los exámenes LPI 101 y 102
- Tests con preguntas tipo de los exámenes
- Artículos relevantes de Linux Magazine
- Ejercicios y actividades prácticas
- Chat y foros para dudas y sesiones con tutores
- Video-tutoriales para cada uno de los puntos
- Clases presenciales en Málaga y Córdoba
- Financiación para empresas y particulares

Linux Professional Institute
aula técnica de Andalucía
Aconsa
LINUX MAGAZINE

Administración de datos en la Shell

BIEN ORGANIZADO

Los comandos adecuados nos ayudan a administrar correctamente ficheros y directorios, algo que puede convertirse en un serio trabajo de reorganización y limpieza. **POR HEIKE JURZIK.**

Si bien los administradores de ficheros gráficos como Konqueror y Nautilus emplean clics de ratón y usan características como las de arrastrar y soltar, también se pueden usar algunos comandos simples en la shell. Este artículo introduce los comandos *mkdir*, *mdir*, *cd*, *touch*, *cp*, *mv* y *rm* y muestra cómo administrar datos de manera altamente eficiente en la consola.

Creación y Supresión de Directorios

El comando *mkdir* ("make directory" o crear un directorio) permite crear una nueva carpeta. Si escribimos

```
mkdir carpeta1
```

se creará un directorio llamado *carpeta1* en el directorio actual. Los privilegios de acceso para el nuevo directorio se definen mediante *umask* (véase el cuadro "umask"). Para asignar y definir diferentes permisos según se desee, puede usarse el parámetro *-m* con un número octal (ver Listado 1).

El comando también comprende valores de rutas absolutos y relativos. Por ejemplo, para crear una carpeta bajo el directorio *musica*, no es preciso cambiar de directorio. En vez de ello, podemos especificar la ruta de la siguiente manera:

```
mkdir musica/Metalica
```

Si el directorio superior no existe, *mkdir* protestará:

```
mkdir: no se puede crear el➥
directorio «musica/Metalica»:➥
No existe el fichero o el➥
directorio
```

En este caso, necesitaremos establecer la opción *-p* para crear jerarquías de carpetas en un único paso. En vez de introducir la lista de comandos siguientes:

```
mkdir musica
mkdir musica/Metalica
mkdir musica/Metalica/Load
```

podría conseguirse el mismo efecto con el parámetro *-p* en un único comando:

Comandos

- mkdir – Permite crear directorios
- rmdir – Permite cambiar directorios
- cd – Ayuda a navegar entre carpetas
- cp – Copia datos
- mv – Mueve cosas
- rm – Proporciona a los usuarios una elegante manera de eliminar ficheros y directorios

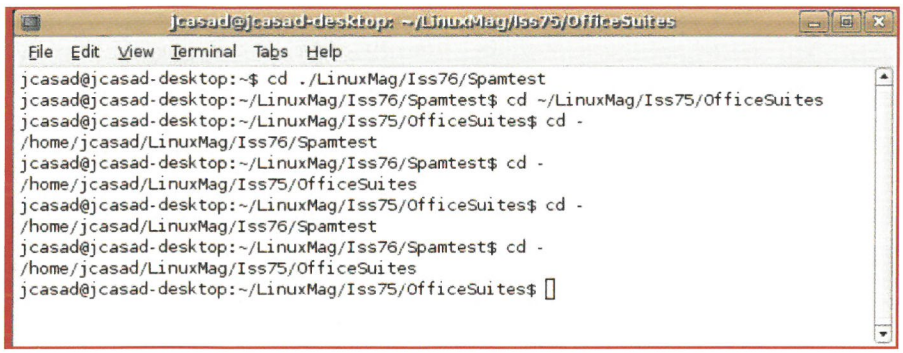


Figura 1: cd lleva al último directorio visitado, y vuelve de nuevo.

```
mkdir -p musica/Metalica/Load
```

Para suprimir un directorio usaremos el comando *rmdir* ("remove directory" o eliminar directorio). Si la carpeta no está completamente vacía, *rmdir* rehusará cooperar:

```
$ rmdir carpeta1
rmdir: «carpeta1/»:
El directorio no está vacío
```

En este caso, podemos eliminar el contenido primero, o bien usar el comando *rm* (véase la sección "Borrón y cuenta nueva"), el cual tiene una opción para forzar la eliminación. *rmdir* también soporta el parámetro *-p* y supone que todas las carpetas están vacías. El comando

```
rmdir -p musica/Metalica/Load
```

eliminará los tres directorios del disco en un santiamén.

El Toque Ideal

El comando *touch* se usa frecuentemente para crear nuevos ficheros vacíos. Si hacemos

```
touch linux
```

se creará un fichero vacío llamado *linux* en el directorio actual siempre que no exista este fichero previamente (en el caso de que sí exista, actualiza la fecha de la última modificación). Este comando también interpreta el *umask* para establecer permisos automáticamente.

Si el fichero existe, *touch* modifica la **marca de tiempo** y establece el último acceso y la última hora en la que se cambió el fichero a la hora actual. Esto cobra mucho más sentido cuando se combina con *make*, por ejemplo. Sin embargo, este comando solamente funciona si han cambiado uno o múltiples ficheros fuente.

Para hacer que *make* se ejecute, incluso si las fuentes no han cambiado, podemos ejecutar *touch* contra los ficheros de código fuente para modificar la marca de tiempo, como en *touch *.txt*, por ejemplo.

Navegación Inteligente

El comando *cd* ("change directory" o cambiar de directorio) permite a los usuarios navegar por las carpetas. Puede especificarse una ruta absoluta o una relativa:

```
cd /var/log
cd ../var/log
```

El comando tiene un par de atajos de teclado prácticos. Si se escribe *cd* sin ningún otro parámetro, nos llevará hasta nuestro directorio de inicio.

cd interpreta la **variable de entorno** *HOME* para descubrir la ruta al directorio de inicio:

```
$ echo $HOME
/home/huhn
```

Bash también usa la variable de entorno *OLDPWD* para guardar el nombre de la última carpeta visitada, y *$OLDPWD* nos vuelve a esa carpeta.

Si escribimos *cd $OLDPWD* una segunda vez, retrocederemos al directorio previo. La shell sólo almacena estos dos directorios, así que esto es lo que se puede retroceder cada vez que ejecuta-

Listado 1: Parámetro -m

```
01 $ mkdir carpeta1
02 $ mkdir -m 777 carpeta2
03 $ ls -l
04 drwxr-xr-x 2 huhn huhn 4096
   2006-12-28 14:07 carpeta1/
05 drwxrwxrwx 2 huhn huhn 4096
   2006-12-28 14:08 carpeta2/
```

umask

Umask define los permisos que el sistema de ficheros asigna a nuevos ficheros y directorios. Puedes escribir umask en la línea de comandos para que presente el valor actual para una variable. La salida será un número octal de cuatro difras que define los permisos que se deniegan.

En nuestro ejemplo, 0022 significa que los ficheros de texto, que típicamente se crean en modo 0666 (accesos de lectura y escritura para todos), se les asigna 0644 (0666 menos 0022), es decir, -rw-r—r—). Para directorios, los permisos por defecto son 0777 (todos los permisos para todos); con un umask de 0022, los directorios pasan a 0755 (drwxr-xr-x).

También se puede usar el comando umask para modificar la máscara en sí. Para que los cambios realizados permanezcan, introducimos el comando .bashrc en el fichero de configuración bajo el directorio de inicio.

Glosario

Marca de tiempo: Los sistemas de ficheros Unix administran un número de entradas de tiempo para un fichero, tales como el último acceso o la hora de la última modificación. Esto es especialmente importante para copias de seguridad, por ejemplo, para mantener la marca de tiempo original (por ejemplo, de la última modificación) para evitar tener que trabajar sobre un fichero erróneo.

Variable de entorno: La shell proporciona a cada usuario espacio en el que almacenar información específica para el acceso mediante programas. Las variables de entorno constan de un nombre y de un valor.

Enlace simbólico: Un indicador a otro fichero que es tratado como este fichero aplicando programas. Si eliminamos el fichero al cual apunta el enlace simbólico o *symlink*, el enlace queda huérfano. Los symlinks se crean mediante el comando *ln -s*.

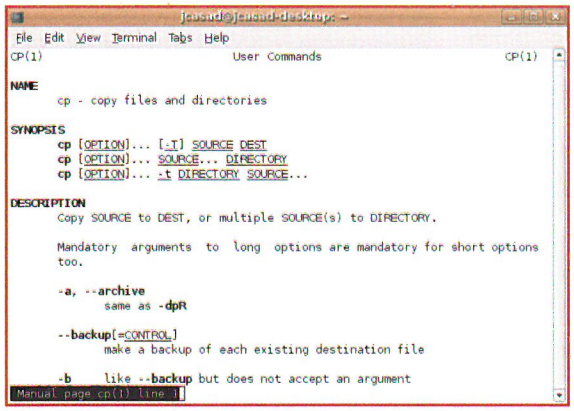


Figura 2: Si se olvidan las opciones de la línea de comandos podemos usar el comando man para ver la página de ayuda.

mos este comando. Es posible guardar algo escribiendo *$OLDPWD* con un signo menos (Figura 1).

Copia de Ficheros y Directorios

El programa *cp* ("copy" o copiar) duplica ficheros. Para usar este comando es necesario especificar la fuente y el destino:

```
cp fichero1 fichero2
```

La copia tiene la marca de tiempo actual y pertenece por defecto al usuario actual. Nuevamente, los permisos se controlan por *umask*. Si se necesitan mantener tantas propiedades del fichero original como sea posible, con el parámetro *-p* haremos que se mantengan los permisos y la marca de tiempo del fichero fuente.

También podemos especificar un directorio como destino. Aunque es posible copiar múltiples ficheros a un directorio, el programa sin embargo no copiará directorios:

```
$ cp carpeta1 carpeta2
cp: se omite el directorio ➥
 «carpeta1»
```

Listado 2: rm

```
01 $ ls -l
02 -r--r--r-- 1 huhn huhn 0
   2006-12-28 16:44 fichero
03 ...
04 $ rm fichero
05 rm: ¿borrar el fichero regular
   «fichero» protegido contra
   escritura? (s/n)
```

Para copiar un directorio hemos de decirle a *cp* que use recursión mediante la opción *-r*. Sin embargo, no es esto todo lo que puede hacer la copia; si se le dice que copie un **enlace simbólico** en lugar de un fichero "normal", *cp* copiará el fichero al cual apunta el enlace. Si en lugar de hacer esto queremos que cree un nuevo enlace deberemos usar el parámetro *-d*.

Hay que tener cuidado si se especifica un fichero con el mismo nombre que el destino, pues *cp* sobreescribirá el fichero ya existente. Podemos añadir una red de seguridad especificando la opción *-i*:

```
$ cp -i fichero1➥
carpeta/fichero1
cp: ¿sobreescribir➥
«carpeta/fichero1» (s/n)?
```

Alternativamente, podemos decirle a *cp* que cree un copia de seguridad mediante la opción *-b*:

```
cp -b fichero1➥
carpeta/fichero1
```

La copia de seguridad tiene una tilde ("~") al final de su nombre de fichero.

¡Vamos mover!

La herramienta *mv* ("move" o mover) puede mover o renombrar ficheros. De nuevo, debemos darle una fuente y un destino al comando. La fuente puede ser un fichero o un directorio, mientras que el destino puede ser un directorio, o el nombre de un fichero o de un directorio. Para cambiar un fichero llamado *fichero1* desde el directorio actual a una carpeta ya existente llamada *directorio1*, simplemente debemos hacer lo siguiente:

```
mv fichero1 directorio1
```

Si la carpeta de destino no existe, este comando cambiará el nombre del fichero a *directorio1*, es decir, deberemos especificar el nuevo nombre para renombrar la fuente.

Al igual que *cp*, *mv* soporta las opciones *-i* y *-b* para proteger los datos contra borrados accidentales mediante la línea de comandos o con la creación de copias de seguridad.

Borrón y Cuenta Nueva

El comando *rm file* elimina el fichero especificado "sin pedir permiso a nadie". *rm* ("remove") no espera y no nos permite ver si realmente estamos seguros. La única excepción sería para el caso en que se posea un fichero de sólo lectura, siendo entonces cuando nos solicita confirmación (ver Listado 2).

Sin embargo, este no es un método muy recomendado. En su lugar, debería usarse la opción *-i*, que cambia el comando al modo interactivo (al igual que hace con *cp* y *mv*):

```
$ rm -i fichero
rm: ¿borrar el fichero➥
regular «fichero»? (s/n)
```

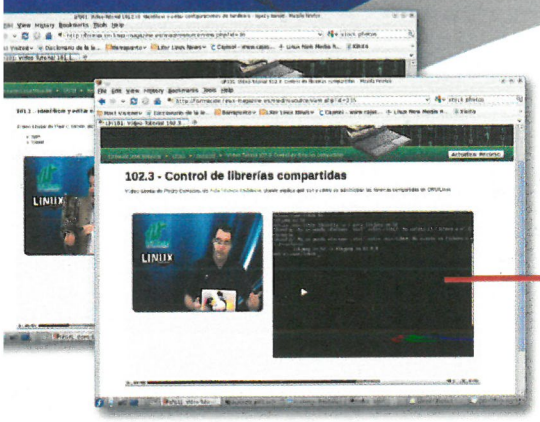
Si se poseen un número elevado de ficheros de escritura protegidos en una carpeta y no nos agrada la idea de tener que confirmar la eliminación de cada fichero, podemos usar el parámetro *-f* ("force" o forzar) para decirle a *rm* que no nos pregunte.

rm también tiene la opción *-r*. Si la configuramos, eliminaremos felizmente subdirectorios junto con sus contenidos de manera recursiva hasta lo más alto de la ruta. Cuando *rmdir* duda si eliminar una carpeta que no está completamente vacía, *rm -r* se encarga. Si se está interesado en ver qué ocurre exactamente, y dónde estuvo *rm*, se aconseja establecer el parámetro *-v* (Figura 2).

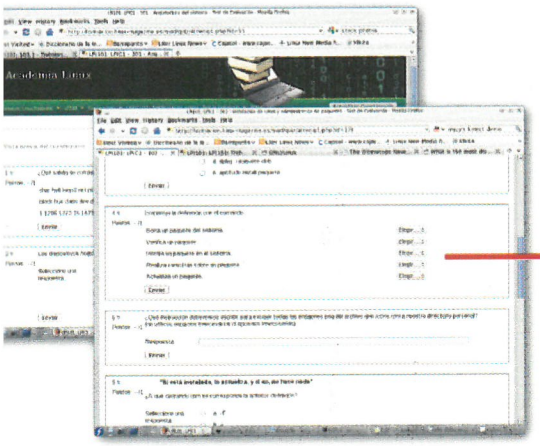
Los ficheros que comienzan con caracteres no estándares, tales como el signo menos, son problemáticos; *rm* (y otros comandos) se negarán a tocarlos. La razón para ello es que la shell interpreta el primer carácter después del signo menos como un parámetro, impidiendo que se encuentre el destino deseado.

Un truco ayuda a resolver esta situación: escribimos *rm ./-fichero* para borrar *-fichero*, impidiendo que se interprete el destino como un parámetro. Como alternativa, podemos insertar dos guiones delante del nombre del fichero (*rm -- -fichero*); de este modo le decimos al comando que lo que sigue a los guiones no es un parámetro, sino un argumento (esto es: el destino de la operación actual). ■

Linux Magazine ACADEMY



Videotutoriales de todos los puntos con explicaciones pormenorizadas de instrucciones y procesos.



Tests de auto-evaluación por apartado y capítulo que te ayudarán a comprender el temario y te preparán para el examen.

Recursos exclusivos Linux Magazine que aportan más información que cualquier libro de texto, con ejemplos sacados del mundo real.

ruébalo gratis en
cademialinux.com

Linux Professional Institute
aula técnica de Andalucía
Aconsa
LINUX MAGAZINE

Find y Locate

OBJETOS PERDIDOS

Los ordenadores actuales con sus múltiples discos duros con Gigabytes y Gigabytes de espacio, almacenan miles de ficheros. Un fichero perdido puede causar gran cantidad de trabajo y también puede plantear un riesgo de seguridad. Afortunadamente, Linux tiene algunas herramientas versátiles para encontrar estos "ficheros perdidos".

POR ANDRÉ SELIG

Si estamos buscando ficheros en Linux, la línea de comandos es la mejor opción para unos resultados rápidos y fiables. Las GUIs como KFind de KDE (Figura 1), están muy bien, pero carecen de ciertas funcionalidades, flexibilidad y velocidad comparables.

El comando más importante para la búsqueda de ficheros es *find*. Sin especificar ninguna opción, la herramienta se limitará a buscar todos los ficheros bajo el directorio actual. Si se quiere buscar en cualquier otro directorio, simplemente hay que especificar el nombre del directorio como primer argumento. Por ejemplo, *find /home* mostrará una lista de todos los ficheros y directorios bajo */home*.

Pero *find* realmente es útil cuando comienza a utilizarse con opciones. El argumento *-name* permite a los usuarios decirle los ficheros a buscar que coincidan con el patrón dado. Por ejemplo, *find ~ -name "*firefox*"* busca todos los ficheros en el directorio home del usuario que incluyan la cadena firefox en sus nombres.

Tiene sentido encerrar los argumentos entre comillas dobles, ya que esto impide que la shell expanda los parámetros como **firefox**.

Opciones, Pruebas y Acciones

find reconoce tres tipos de argumentos: opciones, pruebas y acciones. Las "opciones" influyen en el comportamiento de *find* en sí mismo, por ejemplo, restringiendo el número de niveles de subdirectorios a buscar o restringiendo la búsqueda a una sola partición. Las "pruebas" permiten al usuario restringir la búsqueda a ficheros específicos, por ejemplo, ficheros que tienen menos de una semana, *find -mtime -8*; o ficheros que tienen más de una semana, *find -mtime +7*. Las acciones influyen en la forma que se muestran los resultados. La acción *-exec comando*, por ejemplo, pasaría los nombres de los ficheros encontrados al programa *comando*.

Las acciones también soportan operadores lógicos, permitiendo a los administradores especificar un comportamiento de salidas complejo. Las pruebas pueden, por ejemplo, restringir los comandos a un tipo específico: *-type f* (para buscar sólo ficheros), *-type d* (para buscar sólo directorios) o *-type l* (para buscar sólo enlaces).

Un administrador responsable también debe buscar ficheros que pertenecen a usuarios o grupos inexistentes para proceder a su limpieza. El siguiente comando se ocupará de realizar esa tarea:

```
find -nouser -o -nogroup
```

La opción *-o* es el operador lógico *OR*, que enlaza las dos expresiones, *-nouser* y *-nogroup*. Otro truco muy útil para los administradores es la búsqueda de ficheros por el UID o GID usando

```
find -perm +6000 -ls
```

El *-ls* le dice a este comando que ejecute *ls -lisd* para cada fichero.

Ejecución de Comandos

Como mencionamos más arriba, *-exec* permite usar nombres de ficheros como argumentos y hasta crear pequeños guiones. Pero hay que tener cuidado con los peligros: usar esta funcionalidad como root en directorios públicos (como */tmp* o el directorio */home*) no es muy buena idea. Hay un retardo entre el hallazgo de los archivos que coinciden con el parámetro de búsqueda y la ejecución de la orden indicada, y un atacante ladino podría explotar este hueco. En el pasado ha habido numerosos ataques en instalaciones Unix que utilizaban el comando *find* para limpiar el directorio */tmp*. El

parámetro *-exec* va seguido de un comando y unas llaves ({}). El programa sustituye las llaves por el nombre del fichero que encuentre en la búsqueda. Un punto y coma finaliza el comando. Necesitamos separar el punto y coma con una barra para impedir que la shell lo interprete. El siguiente comando es una variante del comando DOS *ren*, que renombra ficheros con la extensión **.txt* a la extensión **.txt.bak*:

```
$ find . -name '*.txt'➥
-exec mv {} {}.bak \;
```

Si los requisitos de lo que queremos hacer son más complejos, será preferible usar la salida de *find* en un script de la shell. El comando *-printf* puede ayudarnos. El siguiente comando crea un fichero como el mostrado en el Listado 1:

```
$ find /home/mas -type f➥
-name '*.txt' -printf "mkdir➥
-p /export/backup/%h\ncp➥
-p %p /export/backup/%h/➥
%f.copy\n\n"
```

Locate

Además de las características mencionadas aquí, *find* tiene un amplio abanico de opciones que no podemos incluir por cuestiones de espacio. Sin embargo, el programa tiene un gran inconveniente: es lento. *find* analiza los directorios de búsqueda uno por uno y también puede tener que analizar los inodos de los archivos para realizar las pruebas que se le especifican. Si sólo necesita buscar una subsección del directorio *home* con unos cuantos cientos de ficheros, no notará el retardo. Pero si se está buscando un fichero que está oculto en cualquier parte del sistema, *find* necesitará buscar entre los miles y miles de inodos. Naturalmente, esto puede llevar algún tiempo.

locate resuelve este problema mediante la creación de un índice de los ficheros recopilados en el equipo y almacenando sus nombres en una base de datos centralizada. La herramienta no necesita leer los inodos para realizar la búsqueda; en vez de ello buscará en una base de datos. Esta búsqueda normalmente devolverá los resultados en una fracción de segundo, eliminando la necesidad de esperar minutos y evitando también cualquier posible impacto en el rendimiento para usuarios y servicios.

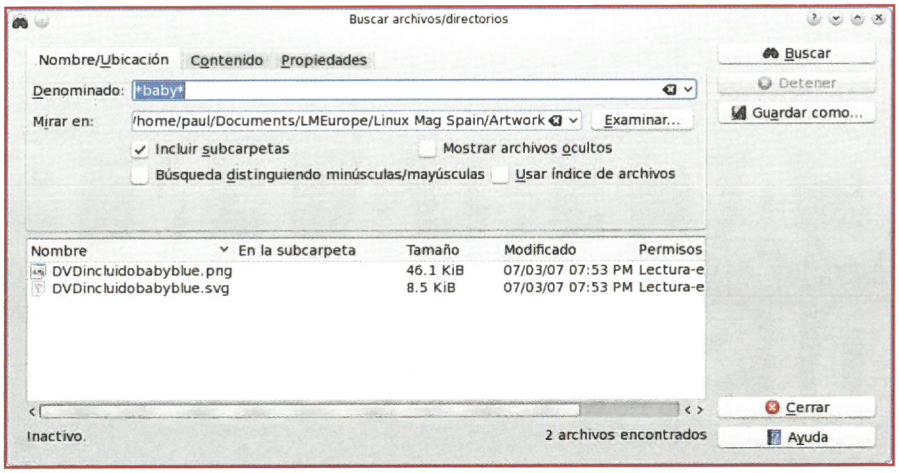


Figura 1: El programa de KDE, KFind, integrado en Konqueror, ofrece una excelente interfaz para la búsqueda de ficheros, pero no puede competir con la flexibilidad de find.

locate tiene unas cuantas limitaciones, así que en muchos casos se sigue utilizando *find*. En vez de la completa sintaxis de *find* en la línea de comandos, *locate* sólo soporta búsquedas básicas de elementos por nombres de ficheros (de hecho, esto es todo lo que se almacena en la base de datos). La herramienta *locate* permite comodines, como la interrogación para un solo carácter y el asterisco para unos cuantos caracteres. Estos comodines también pueden representar al carácter barra (/), a diferencia de lo que ocurre con los comodines de la shell:

```
$ locate /home/mas/*mail*
/home/mas/.fetchmailrc
/home/mas/.procmaildata-bulk
/home/mas/.procmaildata-inbox
/home/mas/.procmailrc
```

Hay dos requisitos básicos para la utilización de *locate*, sin embargo pueden ser bastante molestos. El primero es que no todas las distribuciones de Linux instalan esta utilidad de manera predeterminada. En segundo lugar, la base de datos normalmente se actualiza mediante *cron*, y si se apaga el equipo durante la noche y si la distribución no utiliza *anacron* o una herramienta parecida para actualizar las tareas del *cron* que no se han ejecutado, los usuarios necesitarán actualizar la base de datos manualmente utilizando el comando *updatedb*. ■

Listado 1: Script Generado por find

```
01 mkdir -p /export/backup/
   /home/mas
02 cp -p /home/mas/.emacs
   /export/backup/
   /home/mas/.emacs.copy
03
04 mkdir -p
   /export/backup//home/mas
05 cp -p /home/mas/.fetchmailrc/
   export/backup/
   /home/mas/.fetchmailrc.copy
```

Ficheros de Otros Usuarios

locate utiliza una base de datos centralizada, lo que puede suponer un problema de seguridad en algunos entornos. Si un usuario quiere ocultar un fichero, que normalmente se hace asignándole permisos a tal efecto al directorio (*chmod go -rwx*). Esto hace que los ficheros no sean visibles por los comandos como *find* y *ls*, excepto para los propietarios y, por supuesto, para root.

Sin embargo, si el fichero está almacenado en la base de datos, porque *updatedb* se ejecutó con privilegios de root (mediante *crontab* o por root), serán visibles para todo el mundo. Eso sí, los otros usuarios sólo conseguirán ver el nombre del fichero.

Los sistemas multiusuario pueden utilizar los parámetros *--localuser* y *--netuser* cuando realicen la llamada a *updatedb*. La herramienta espera un nombre de usuario como argumento. El nombre de usuario debe ser *nobody*, por ejemplo, en un sistema en el que los usuarios tengan privilegios limitados. Esto significa que la base de datos de *locate* sólo contendrá un mínimo de ficheros visibles.

Búsqueda de archivos de texto con grep

CACERÍA EN EL DISCO DURO

Los eruditos de la Edad Media hubiesen vendido sus almas a cambio de la ingente cantidad de literatura que abunda actualmente en Internet. Hoy en dia, los documentos desordenan nuestros discos duros debido a las descargas indiscriminadas. ¿Cómo encontramos exactamente un pasaje de un texto en nuestra base de datos digital? El comando del shell grep nos puede ayudar a encontrar esa cita escurridiza.

POR ELISABETH BAUER

Las cosas que no seamos capaces de recordar las debemos guardar en nuestro ordenador, reza una máxima del usuario informático.

Ésta no es una mala idea, pero, a diferencia de lo que ocurre con la memoria humana, que normalmente recuperará la información almacenada de forma fidedigna (excepto en los exámenes finales, por supuesto), no siempre es tan fácil encontrar información en nuestro disco duro. Podemos perder mucho tiempo en encontrar un archivo del que hemos olvidado su nombre o dónde fue almacenado. Incluso el saber exactamente qué archivo contiene la información que buscamos puede ser de poca ayuda en caso de archivos de texto grandes.

El comando del shell grep, que localiza cadenas de texto en ficheros, es útil en ambos casos. En la situación más simple, podemos ejecutar grep con la tecla de búsqueda y el archivo a buscar. grep nos mostrará todas las líneas en el archivo especificado que contengan el texto buscado. Imaginemos que deseamos buscar en el archivo *biblia.txt* el texto "Jardín del Edén". Debemos escribir

```
grep Edén biblia.txt
```

en la línea de comandos y grep nos mostrará los pasajes apropiados del archivo.

Si la búsqueda contiene espacio debemos utilizar comillas. Por ejemplo:

```
grep "Jardín del Edén" ➥
biblia.txt
```

Debemos prestar atención y esperar fallos cuando usemos caracteres espciales: *, ? y ! tienen un significado especial para la línea de comandos. Otro grupo de caracteres (., *, ^, $ y \) no se interpreta por grep tal cual. En su lugar, la herramienta supondrá una expresión regular. El lado positivo es que esto nos permite construir búsquedas muy potentes y complejas, si bien es posible que prefiramos evitar estos caracteres con grep hasta que nos sintamos a gusto con la herramienta.

Si no sabemos que archivo contiene el texto que buscamos, ejecutamos grep con un comodín. La obra "Moby Dick" de Herman Melville está compuesta de una colección de archivos de texto.

```
grep blanco moby.*
```

nos mostrará todas la coincidencias con la palabra blanco que aparecen en esta obra maestra de Melville (ver Figura 1). El asterisco significa "cualquier grupo de letras". El interfaz cambiará esta expresión por el nombre de cualquier archivo en el directorio actual que empiece por los caracteres moby.

Si los archivos en los que deseamos buscar están distribuidos por diversos directorios por el disco duros, debemos añadir a la línea la opción r para indicarle a grep que busque en una carpeta completa:

```
grep -r blanco Melville/obras/
```

buscará blanco en el directorio obras y en todos los subdirecorios que cuelguen de él.

Grep de un Vistazo

Comando	Acción
grep patrón fichero	búsca un patron en un fichero
*grep patrón *.htm*	busca en todos los archivos en un directorio que acaban con el sufijo .htm
grep -r patrón dir	realiza una búsqueda recursiva en un directorio y sus subdirectorios.
grep -i patrón fichero	ignora la diferencia entre fichero mayúsculas y minúsculas.
grep -A n	muestra las siguientes n líneas tras la línea que contiene la búsqueda.

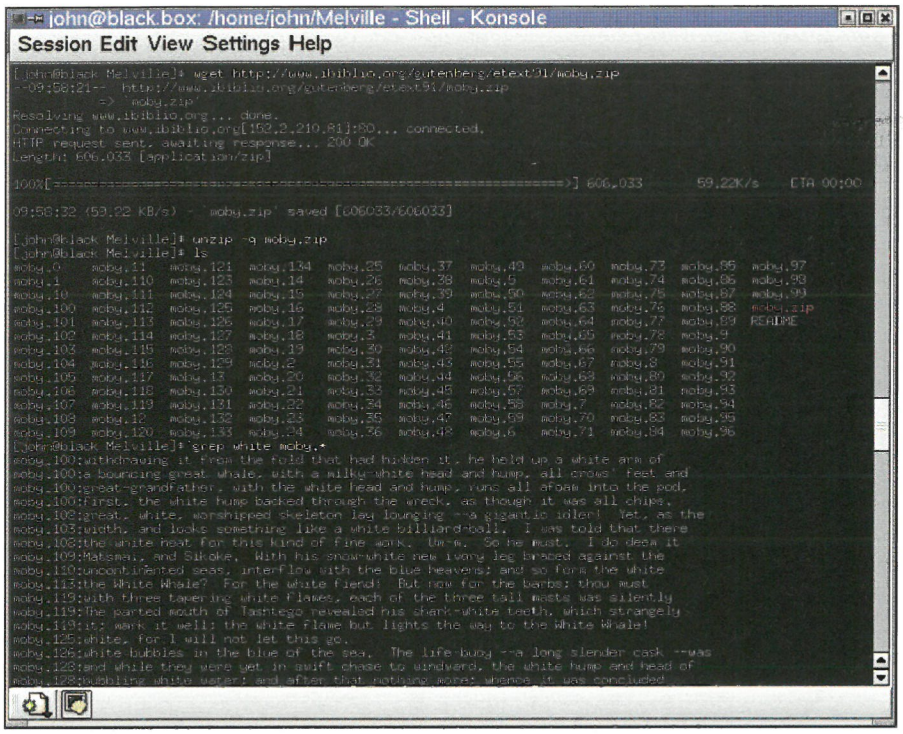


Figura 1: El libro "Moby Dick" de Melville está subdividido en una serie de archivos de texto. El comodín "moby.*" indica a grep que busque en todos los archivos del directorio.

El Trio: ps, grep y kill.

grep no es sólo útil para búsquedas de texto filosóficos y teológicos, sino que puede ser combinado con otros comandos del shell. Si la respuesta de un comando produce mucho texto, grep puede ser utilizado escribiendo tras la instrucción el carácter "|" y *grep texto_a_buscar* para filtrar el resultado y mantener sólo las partes en las que estamos interesados. Un caso típico es aquel en el que usaríamos grep para cerrar un programa que se ha bloqueado.

El comando *ps ax* nos muestra los procesos activos. Podemos usar grep para aplicar un filtro y encontrar sólo el programa que estamos buscando, *Mozilla* por ejemplo:

```
> ps ax | grep mozilla
2500 ? S 1:40 /usr/lib/➥
mozilla-1.3/mozilla-bin
5645 pts/4 S 0:00 grep mozilla
```

grep nos muestra dos coincidencias que contienen *Mozilla*: el buscador y el propio grep. La parte que nos interesa para cerrar el programa aparece al principio de cada línea: el ID del proceso. Ahora podemos escribir *kill 2500* para cerrar el programa deseado.

Debido a que los gurús del shell suelen ser notablemente perezosos, necesitamos encontrar un método para no tener que escribir este comando cada vez que lo requiramos: en otras palabras, necesitamos un *alias*. Los alias definidos deben ser guardados en el archivo *.bashrc* en el directorio raíz. Este archivo se ejecuta cada vez que abrimos un shell de comandos interactivo. Usaremos nuestro editor de texto favorito para abrir el fichero, por ejemplo kwrite *~/.bashrc &* o *vi ~/.bashrc* en el escritorio, dependiendo de nuestras preferencias. Debido a que vi no es sencillo, veamos algunos comandos simples. Si escribimos *G*, le indicamos a *vi* que debe ir al final del archivo. Entonces podemos escribir *o* para cambiar el editor al modo de introducción de datos. A diferencia de los comandos *a* e *i*, *o* indica a *vi* que inicie la inserción en la línea donde está situado el cursor.

Ahora podemos introducir nuestro alias en la última línea de *.bashrc*. En lugar de *pss* podemos usar cualquier otro nombre fácil de recordar, pero evitando usar el nombre de un comando existente:

```
alias pss="ps ax |➥
grep"
```

Ahora presionamos [Esc] *ZZ* o [Escape] *:wq* para almacenar el archivo y salir de *vi*. Para utilizar nuestro nuevo alias en el shell necesitamos analizar sintácticamente el archivo de configuración. Para hacerlo debemos escribir:

```
. ~/.bashrc
```

Ahora podemos usar el comando *pss nombreprograma* para buscar un programa activo.

Libro de Direcciones

grep es una herramienta tremendamente flexible. Una de mis aplicaciones favoritas para grep es un sencillo libro de direcciones.

Todo lo que necesitamos para nuestro "programa" son los comandos *grep*, *alias* y *cat* y un fichero de texto donde hayamos ido almacenando los nombres, números de teléfono, direcciones postales y las direcciones de correo electrónico de nuestros amigos, conocidos y parientes. Una entrada podría tener el siguiente aspecto:

```
Charly Pingüino
+12345 678
tux@linux.org
C/del Polo Sur
Villatux, Antártica
```

Ahora debemos guardar el archivo bajo el nombre *direcciones* en nuestro directorio raíz y añadir el siguiente alias a *.bashrc*:

```
alias tel="cat ~/direcciones➥
 | grep -i -A 4"
```

El comando *cat* nos muestra el contenido del archivo direcciones. El carácter | envía este resultado a grep. La opción *-i* garantiza que la búsqueda no diferenciará entre mayúsculas y minúsculas. Finalmente, *-A 4* indica a grep que debe mostrar las 4 líneas siguientes a la primera coincidencia con la búsqueda.

De nuevo

```
. ~/.bashrc
```

analizará sintácticamente nuestro archivo de configuración. En el futuro, sólo necesitaremos escribir tel nombre en la shell para recuperar la dirección de la persona que deseemos buscar. ■

Redireccionamiento de entradas y salidas

FLUJO DE DATOS

www.sxc.hu

Trabajar con el shell tiene muchas ventajas. Tuberías, redireccionamientos y cadenas de comandos proporcionan a los usuarios casi un número infinito de opciones. **POR HEIKE JURZIK**

La combinación de los comandos del shell nos permite incrementar el poder de la línea de comandos. Esto no sólo incluye el enlace a programas individuales, sino también la transferencia de la salida de comandos. Por ejemplo, si la información que necesitamos desaparece de la pantalla tan rápidamente que nos impide leerla, podemos redireccionarla a un fichero o usar un paginador.

Para enlazar comandos individuales en bash sólo son necesarias un par de cosas. En vez de decir

```
$ mkdir carpeta
$ cd carpeta
$ cp ../carpeta2/* .
```

podemos combinar estos pasos y usar punto y coma para decirle al intérprete que ejecute estos comandos en una sucesión rápida:

```
$ mkdir ➱
carpeta; cd carpeta; ➱
cp ../carpeta2/* .
```

Para ejercer mayor control sobre bash, podemos pasarle una condición. Por ejemplo, podríamos ejecutar un segundo comando (o tercero, o enésimo) sólo cuando el comando anterior se haya completado exitosamente o si ha fallado.

Para asegurarnos de que un fichero existe antes de borrarlo, añadiríamos una condición de prueba simple a la línea de comandos con *rm*:

```
test -w fichero && rm fichero
```

El programa test simplemente comprueba para ver si un fichero existe y, en este caso, si es de lectura-escritura (parámetro *-w*), en cuyo caso *rm* lo borraría. Si necesitas compilar software frecuentemente desde el código fuente usando el grupo normal de comandos *./configure*; *make*; *make install*, puedes combinar los tres usando && para asegurarte de que cada comando sucesivo sólo se ejecuta si el previo no tenía ningún error:

```
./configure && make && ➱
make install
```

Además de la opción *-w*, el comando *test* dispone de otros prácticos usos. Por ejemplo, *-d* permite comprobar si un directorio existe; si no fuera así lo podemos crear:

```
test -d carpeta || mkdir carpeta
```

Los dos caracteres tubería entre ambos comandos actúan como un OR lógico.

Para la entrada y salida de los comandos en el shell no existen más que tres canales: Los programas leen datos desde su entrada estándar (STDIN, canal 0) o desde un fichero, la salida del programa se envía a una salida estándar (STDOUT, canal 1) y los mensajes de error se escriben en la salida estándar de error (STDERR, canal 2):

```
comando < entrada > salida 2> ➱
error
```

Los operadores < y > indican la dirección: si la entrada estándar no tiene su origen en el teclado, < le dice a *comando* que lo lea desde un fichero. Para redireccionar la salida a un fichero necesitas el operador >. La salida de error también usa >; sin embargo, se debe especificar el canal añadiendo una descripción del fichero (2>).

Redireccionando la Salida

Como he mencionado antes, el operador > redirecciona la salida desde un programa a un fichero. En lugar de > también podríamos escribir *1*> porque la opción especifica primer canal; sin embargo, esto no es estrictamente necesario, porque el shell supondrá que tiene que enviar los datos a la salida estándar si no le dices lo contrario:

```
ls /etc >_contenido.txt
```

Si el fichero detrás del operador ya existe, el shell lo sobreescribe. La condición test que examinamos antes previene que esto ocurra:

```
test -w etc_contenido.txt ||
ls /etc > etc_contenido.txt
```

O puedes utilizar el operador > >:

```
ls /etc >> etc_contenido.txt
```

Los dos signos "mayores que" (> >) significan que el shell ha de añadir la salida desde el comando *ls* al fichero *etc_contenido.txt* si este fichero ya existe. Si no, el shell crearía un nuevo fichero y escribiría la salida en dicho fichero.

Captura de Mensajes de Error

Para redireccionar el segundo canal usa el número 2 seguido del operador >. Este método es idóneo si un programa genera tantos mensajes de error que nos evita leer la salida actual del programa:

```
$ find /home -name "*.tex"
find: /home/lost+found:
No permissions
find: /home/petronella/data:
No permissions
/home/huhn/book/book.tex
/home/huhn/book/chap01.tex
...
```

El comando

```
find /home -name "*.text" 2>
/dev/null
```

permite enviar mensajes de error a */dev/null* de nuestra máquina, evitando que se sature la salida estándar.

Dos Pájaros de un Tiro

Una combinación inteligente de operadores nos permite redireccionar dos canales al mismo tiempo. Si deseas escribir la salida estándar a un fichero desde el comando *find* en el ejemplo anterior, aunque sin que se registren todos los mensajes de error, usa:

```
find /home -name
"*.text" > 
salidafind 2> 
/dev/null
```

El doble operador > >, que creará ficheros que no existen o los añadirá a ficheros que sí existen, también es útil en este caso. El ejemplo anterior muestra cómo hacer esto para una salida estándar. Para el caso de salidas estándar de errores podemos usar la doble flecha con el mismo efecto si necesitamos escribir mensajes de error a un fichero:

```
find /home -name "*.text" > 
salidafind 2>> error
```

Fontanería

Las tuberías a menudo nos evitan pasos al direccionar la salida directamente desde un programa a otro sin que sea necesario el desvío a través de un fichero. El carácter pipe o tubería (|) separa los comandos individuales, como se demuestra en el siguiente ejemplo.

```
ls /etc | less
```

Esto envía la salida desde el comando *ls* al paginador *less* o directamente al terminal. El paginador muestra la salida pantalla por pantalla y soporta el desplazamiento con las teclas del cursor. La tubería es muy común en combinación con salidas de *grep* para flujos específicos; por ejemplo,

```
find debian -name "*.png" | 
grep –color apt
```

busca en el directorio *debian* todos los ficheros que acaban en *.png* y pasa la salida a *grep*. Grep busca en el flujo la secuencia de caracteres *apt* y destaca los que encuentra en rojo gracias a la opción *--color* (Figura 1).

Puedes usar distintas tuberías. El comando siguiente lista el contenido de tu directorio de inicio línea a línea, pasa los resultados a la herramienta *grep*, busca en el flujo *.jpg* y cuenta las coincidencias (a efectos prácticos: cuenta el número de ficheros de imagen JPEG en tu directorio de inicio):

```
$ ls -l ~ | grep .jpg 
| wc -l 12
```

Cuando usé este comando encontré 12 ficheros JPEG en mi directorio de inicio sin tener que hacer nada muy enrevesado y utilizando herramientas estándar de la línea de comandos.

La Hora del Tee

Para desviar la salida de un comando a más de un sitio usamos el programa Tee. El comando espera datos desde la entrada estándar y los puede escribir simultáneamente tanto a un fichero como en pantalla.

Podemos colocar una "Tee junction" (o intersección Tee) al final de la línea de comandos o entre comandos individuales:

```
comando1 | tee salida.txt |
comando2
```

Queremos, por ejemplo, buscar ficheros PNG en el directorio actual y sus directorios hijos y pasar la salida al fichero *imagenes.txt* a la vez que presentarla en pantalla. Simultáneamente queremos que *grep* busque la cadena *libro* en el nombre de las imágenes. Para ello, escribiremos:

```
find . -name "*.png" | 
tee imagenes.txt | grep libro
```

Por defecto, Tee sobreescribirá el fichero si existe. Para añadir la salida a un fichero existente, simplemente añadimos el parámetro *-a*:

```
find . -name "*.png" | tee 
-a imagenes.txt | grep libro
```

Los distintos operadores, junto con la tubería y los comandos Tee soportan combinaciones de comandos extremadamente flexibles. No vale la pena crear un script simplemente para búsquedas rápidas, por ejemplo.

Pronto descubrirás incluso nuevos usos y nuevas combinaciones. ■

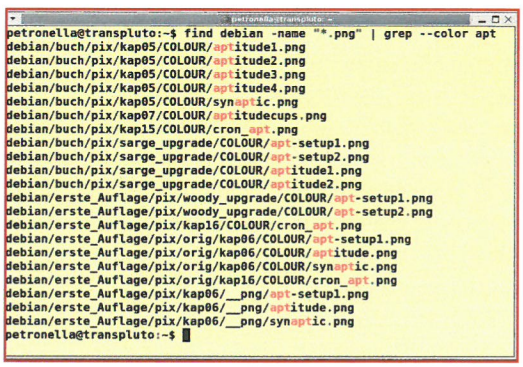
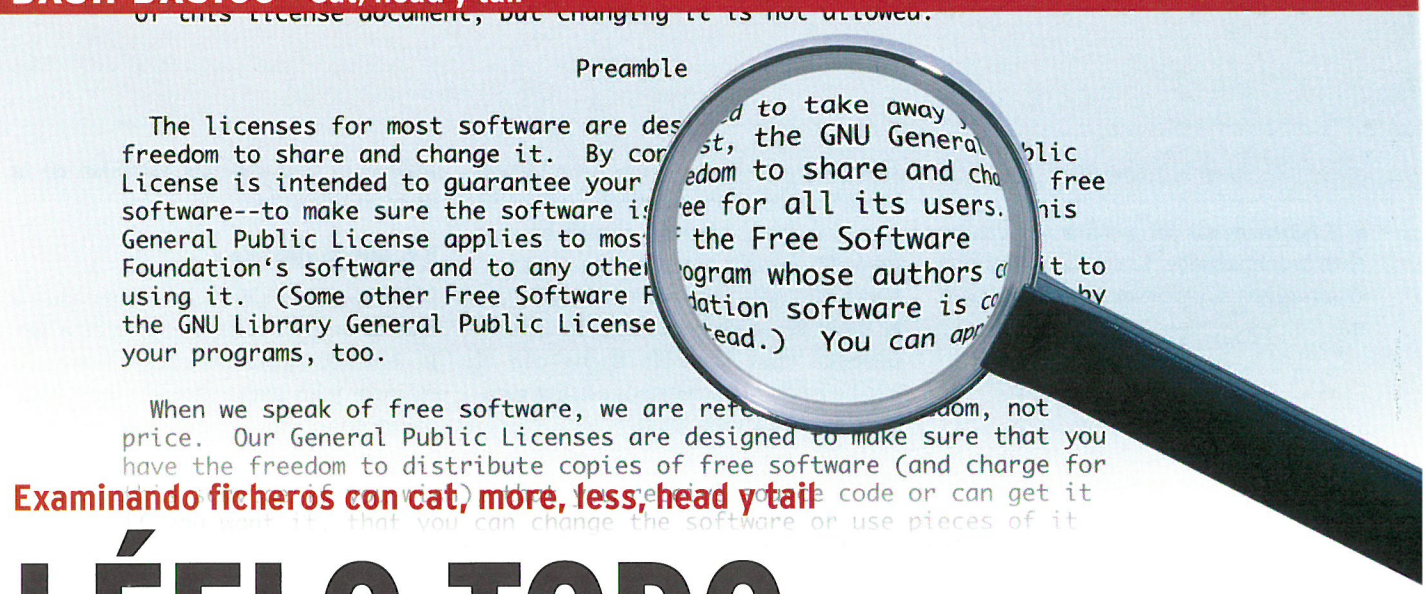


Figura 1: La salida del comando find se envía a la herramienta grep sin desviarla a través del shell.



Examinando ficheros con cat, more, less, head y tail

LÉELO TODO

Herramientas prácticas como cat, less, y head son ideales para visionar textos. **POR HEIKE JURZIK**

No siempre es necesario arrancar un editor para visionar los ficheros de texto en la consola. Muy a menudo, un simple visor es mucho más rápido. Además, los programas que vemos en el número de este mes tienen mucho más que ofrecer. Si se está buscando un método para concatenar múltiples ficheros en uno solo, cat es la herramienta apropiada. Este artículo también examina cómo funcionan para los paginadores *more* y *less*, enseñando los métodos más convenientes para el uso de estas herramientas. Y si se está interesado en coger las primeras o las últimas líneas de un fichero, vemos cómo hacerlo con *head* y *tail*.

El Gato con Botas

cat permite mostrar un fichero de texto en la salida estándar:

```
cat fichero.txt
```

Para añadir números de línea a esta salida, podemos escribir *-n*. Si se pretende mostrar con *cat* un fichero binario en lugar de uno de texto, la shell presentará todo tipo de extraños caracteres. Para resetear la pantalla se escribe *reset* en la línea de comandos (puede que tenga que hacerlo a ciegas).

Pero *cat* puede hacer mucho más. El nombre del programa deriva del verbo inglés "to concatenate" (es decir, concatenar), ya que *cat* realmente lo que permite es pegar juntos dos o más ficheros. Para concatenar dos ficheros de texto en uno se usan *cat* y el operador >:

```
cat fichero1 fichero2 > fichero3
```

Se aconseja ser prudente si ya existe el *fichero3*, ya que *cat* lo reescribirá. Pero puede usarse el operador > > para decirle a *cat* que adjunte los dos primeros al tercero sin sobreescribir su contenido:

```
cat fichero1 fichero2 >> fichero3
```

Más o Menos

Si necesitamos ver un fichero de texto en la shell, pero permitiéndonos mover arriba y abajo y buscar en el fichero, lo mejor es usar un paginador como *more* o *less*. Para ver el fichero página a página se escribe:

```
more fichero
```

Se presiona la barra espaciadora para bajar una línea y la tecla Enter para subir otra. Una línea de estado en la parte inferior de la página muestra la posición de la página actual a modo de un porcentaje del fichero completo. [B] ("back") lleva a la página anterior, [H] presenta la ayuda y [Q] ("quit") sale del programa. *more* sale automáticamente cuando llega al final del fichero.

Además, *more* posee unos cuantos atajos de teclado más: si se está interesado en el número de la línea actual, se pulsa la tecla del signo igual (=), la combinación de [Shift + ;] y [F] (*:f*) presenta el nombre del fichero con el número de la línea. Para buscar en el fichero, se escribe el carácter barra (/) y se introduce la clave de búsqueda.

Si fuera necesario, *more* puede presentar múltiples ficheros de texto simultáneamente. Para hacerlo, simplemente es suficiente pasar todos los nombres de los ficheros a la herramienta cuando ésta se arranca. El paginador indica el nombre de cada fichero:

```
::::::::::::::
wget.txt
::::::::::::::
```

Puede presionarse *:n* ("next", es decir, siguiente) o *:p* ("previous" o previo) para saltar de un fichero a otro. El comando *:p* lleva al comienzo del fichero actual; si se presiona *:p* por segunda vez, vuelve al fichero de texto previo.

Para editar un fichero no es necesario dejar *more* y arrancar tu editor favorito. En lugar de eso, se presiona [V] mientras se está viendo el fichero. Con ello se arranca por defecto el editor vi, pudiéndose entonces modificar el fichero. Si se prefiere usar *more* para invocar a un editor diferente hay que modificar el valor de la variable de entorno *EDITOR*, la cual es evaluada por otros muchos programas. Por ejemplo, si se prefiere trabajar con el editor Xemacs, puede definirse la variable con el siguiente comando:

```
export EDITOR=xemacs
```

Cuando se sale del editor, se vuelve a *more*. Para modificar permanentemente el valor de la variable de entorno *EDITOR*, se añade el comando *export* al fichero de configuración bash, *~/.bashrc*, y se reanaliza el fichero como sigue:

```
source ~/.bashrc
```

Las opciones permiten influir en el modo en el que reacciona *more*; éstas pueden ser especificadas cuando se arranca la herramienta. Por ejemplo, si se necesita editar un fichero con muchas líneas vacías, puede introducirse

```
more -s fichero
```

para decirle al paginador que reduzca múltiples líneas vacías a una sola. Para restringir la salida a 10 líneas en la ventana actual, se añade *-10* como una opción de arranque, y para evitar que *more* se olvide de esta configuración cuando se desplaza a través del fichero, se añade la opción *-p*.

Si se desea que *more* use los mismos parámetros de la línea de comandos cada vez que se ejecute la herramienta, puede establecerse la variable de entorno *MORE* y conservar mucho de lo escrito. Para hacerlo se añade lo siguiente (o algo similar) al fichero de configuración bash *~/.bashrc*:

```
export MORE=" -10 -p -s"
```

seguidamente, para reanalizar el fichero ejecutamos el siguiente comando:

```
source ~/.bashrc
```

Menos es Más

less es el sucesor de *more*: reconoce los mismos atajos de teclado, aunque añade unas cuantas funciones propias. Si no se desea usar la barra espaciadora y [B] para moverse, pueden emplearse los cursores en su lugar. [G] salta al comienzo del fichero y [Shift + G] al final. *less* no solamente dispone de una búsqueda hacia adelante (/), sino que con el interrogante (?) podemos movernos hacia atrás.

A diferencia de *more*, este paginador no sale automáticamente cuando se alcanza el final del fichero, sino que es preciso salir explícitamente del programa escribiendo [Q]. Mientras que *more* no limpia la presentación cuando sale y presenta el último trozo de texto en la pantalla, *less* la deja limpia. Este comportamiento puede cambiarse si se desea estableciendo el parámetro *-X* en la línea de comandos. Al igual que en *more*, puede definirse una variable de entorno para *less* y así especificar los parámetros por defecto. Por ejemplo, para usar la opción *-X* permanentemente hay que añadir la siguiente línea a *~/.bashrc*:

```
export LESS=" -X"
```

Como en *more*, *less* analiza la variable EDITOR y arranca el editor de texto cuando se pulsa [V]. Los atajos de teclado que se usan para visionar múltiples ficheros son idénticos. Si se pasan múltiples ficheros al paginador, puede presionarse *:n* para moverse al fichero siguiente y *-p* para volver al anterior. A diferencia de *more*, *less* recuerda la posición en el fichero cuando se escribió el comando.

Otra característica práctica del comando *less* es que, si se está trabajando con múltiples ficheros, muestra el nombre del fichero siguiente en la línea de estado de la parte inferior de la página:

```
textfiles.txt (file 1 of 2) ?
(END) - Next: spellcheck.txt
```

Para evitar tener que presionar *:p* para cambiar al fichero previo, puede establecerse la opción *-e* cuando se arranca *less*. Con esta opción se le dice que cambie automáticamente al fichero siguiente cuando llegue al final del actual y se pulse la barra espaciadora, [Cursor hacia abajo] o [Avance de Página].

less dispone de una característica práctica para controlar los cambios de los ficheros. Si se presiona [f], el paginador muestra las líneas añadidas recientemente. Esto resulta extremadamente práctico mientras se usa *less* para ver ficheros log. La línea de estado presenta *Waiting for data...(interrupt to abort)* para mostrar que *less* está esperando cambios (Figura 1). [Ctrl + C] sale del modo "follow" y vuelve a las funciones de paginador normal.

Paquetes Empaquetados

Los comandos *zless* y *zmore* hacen posible que se vean los ficheros comprimidos o zipeados. Las herramientas ofrecen los mismos atajos de teclado que los paginadores de los que heredan. En lugar de usar una secuencia de comandos complicada como:

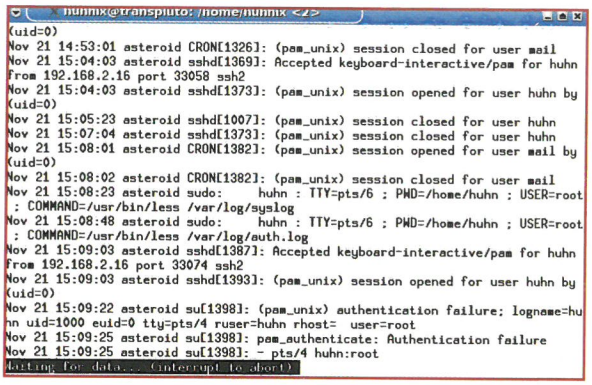


Figura 1: Pulsa [F] para que less muestre nuevas líneas de un fichero.

```
zcat Xinerama-HOWTO.gz | less
```

puede escribirse

```
zless Xinerama-HOWTO.gz
```

para leer el fichero comprimido en la pantalla.

De Cabo a Rabo

Si se está interesado en las primeras o en las últimas líneas de un fichero de texto, *head* y *tail* son las herramientas que necesitas:

```
head fichero.txt
```

presenta las primeras diez líneas de un fichero.

tail es el equivalente de *head* para el final de fichero. Al igual que *head*, puede manejar como entrada múltiples nombres de ficheros.

tail posee muchos extras prácticos: *-f* es con toda probabilidad el más comúnmente utilizado. Al igual que ocurriera con el paginador *less*, esta opción permite que se active el modo "follow" y se registre cualquier cambio. Esto es útil para controlar los ficheros log. El comando siguiente:

```
tail -f /var/log/messages
```

envía a *tail* a un bucle infinito, donde controla un fichero en busca de cambios y presenta las últimas entradas. Para abandonar el modo "follow" se presiona [Ctrl + C]. ■

Búsquedas en tablas de texto

AWK

© James Steidl - Fotolia.com

AWK permite realizar sencillas búsquedas y modificaciones en ficheros de texto formateados como tablas. Los administradores sienten una especial predilección por este lenguaje, que se encuentra instalado por defecto en la mayoría de los sistemas Unix.

POR NICO HOCHGESCHWENDER

La manipulación automática de ficheros de texto forma parte del día a día de cualquier administrador de sistemas. Esta tarea tan importante puede implicar evaluar ficheros de registro, crear o modificar configuraciones o añadir nuevas cuentas.

Para algunas tareas, el clásico lenguaje de programación de Unix, AWK, ofrece la solución más eficiente. AWK es un lenguaje compacto con una sintaxis similar al C, lo que lo hace sencillo de aprender para aquellos con conocimientos de C.

Un script AWK analiza un fichero de entrada línea a línea, buscando pautas. Cuando AWK encuentra una coincidencia, realiza una acción preestablecida. Si el programador no especifica una pauta, AWK simplemente realiza la acción para todas las líneas. En este artículo aprenderemos que AWK es una herramienta muy eficiente para las búsquedas en cualquier tipo de fichero de texto, incluyendo aquellos preformateados en forma de tabla. Un fichero de texto formateado, junto con unas sencillas instrucciones AWK, pueden servir como un simple y flexible sistema de recuperación de datos sin las complicaciones de una base de datos SQL. Este artículo describe cómo un administrador de sistemas puede utilizar AWK para obtener información sobre ordenadores en una red local.

La versión libre (es decir, liberada bajo GPL) de AWK se llama gawk [1] y es un componente estándar de cualquier distribución Linux. Y, debido a que los sistemas tradicionales Unix también incorporan AWK, es una herramienta útil para programación multiplataforma. Si se dispone de servidores Solaris, HP/UX y AIX, además de máquinas Linux, AWK puede ser una herramienta indispensable.

Los ejemplos de este artículo se basan en la lista de ordenadores del Listado 1. La tabla muestra los contenidos de un fichero de texto que lista el nombre del ordenador, su dirección IP, el sistema operativo, el software y su RAM. La idea es que cuando se reconfigura o añade un ordenador, el administrador actualice esta lista.

Método

La sintaxis básica para un script de una sola línea gawk es como sigue:

```
gawk [opciones] ➥
'programa AWK' ➥
fichero_de_entrada
```

Los guiones AWK que pasen de una línea de longitud, por comodidad y claridad, deberán guardarse en un fichero separado. En ese caso, la sintaxis para su ejecución es *gawk [opciones] -f guion.awk fichero_de_entrada*. Lo primero que queremos intentar es que AWK nos proporcione una lista de los nombres de los ordenadores en nuestro fichero de ejemplo del Listado 1:

```
gawk '{print $1}' lista
```

El campo *$1* se refiere a la primera columna. Si se necesita saber la dirección, podemos reemplazar *$1* por *$2*. *$0* corresponde a toda la línea. De hecho,

```
gawk '{print $0 }' lista
```

imprime todo el fichero en pantalla.

Ejemplos

La clave de búsqueda siguiente nos mostrará toda la información para un ordenador llamado *Goofy1*:

```
gawk '$1=="Goofy1" ➥
{print $0}' lista
```

En cada línea, AWK comprueba si ocurre la expresión *Goofy1* en la columna *$1*. Si es así, imprime toda la línea ({*print $0*}). También podemos utilizar en vez del operador de igualdad (= =), el operador de negación (*!* = - ver Tabla 1) para conseguir el efecto contrario, es decir, listar todas las líneas excepto la línea de *Goofy1*.

AWK también puede buscar en rangos, aparte de buscar resultados individuales. El ejemplo siguiente utiliza dos expresiones regulares (ver Tabla 2) delimitado por barras. AWK compara toda la línea.

```
gawk '/Goofy1, /Asterix1/ ➥
{print $0}' lista
```

La salida comprende todos los registros desde *Goofy1* hasta *Asterix1*, ambos inclusive.

Listado 1: El Fichero "lista"

DagobertDuck	10.1.1.3	Debian	Kylix	256
Goofy1	10.1.1.4	Solaris	Mathematica	512
MickeyMouse	10.1.1.5	Debian	Apache	512
LuckyLuke1	10.1.1.6	Debian	Samba	256
LuckyLuke2	10.1.1.7	Debian	Eclipse	256
LuckyLuke3	10.1.1.8	Suse	Mupad	256
LuckyLuke4	10.1.1.9	Debian	Mupad	128
LuckyLuke43	10.1.1.10	Debian	Mupad	128
LuckyMickeyMouse	10.1.1.1	Debian	Mupad	128
Asterix1	10.1.1.12	RedHat	NetBeans	128
Asterix2	10.1.1.13	Debian	NFS	256
Obelix 1	0.1.1.14	RedHat	ICC	256
Apple1	10.1.1.15	OSX	Photoshop	1024
Apple2	10.1.1.16	OS6	Photoshop	128
Apple3	10.1.1.17	OSX	Photoshop	512

Operadores Lógicos

La claves de búsqueda pueden extenderse y enlazarse utilizando operadores booleanos (Tabla 1), tales como el operador *AND*.

```
gawk '($3=="OSX") && ➥
($4=="Photoshop") ➥
{print $1}' lista
```

Los operandos lógicos se han de colocar obligatoriamente entre paréntesis y esto puede dar lugar a errores y a cierta confusión, sobre todo si se utilizan expresiones más complejas. AWK también dispone de un operador lógico *OR* (||) además del *AND* lógico.

Salida

Hasta el momento nos hemos contentado con enviar la salida a la pantalla, pero algiaul que lo que ocurre en la shell, no tiene ningún problema para AWK redirigir flujos de datos a ficheros:

```
gawk '$1=="Obelix" ➥
{print $0 > "/home/ ➥
linux/test"}' lista
```

Tabla 1: Operadores AWK

Operador	Significado
$	Operador de campo
-++	Incremento y decremento posterior
++-	Incremento y decremento anterior
^	Elevar a (potencias)
!	Negación lógica
+-	Operaciones de signos
*/%	Operaciones de multiplicación, división y resto
+-	Suma y resta
<	Menor que
<=	Menor o igual a
==	Igual a
!=	Distinto a
>=	Mayor o igual a
>	Mayor que
~ !~	Compara con expresiones regulares
&&	Y lógico
||	O lógico
=	Asignación
+=	Suma y asignación
-=	Resta y asignación
*=	Multiplicación y asignación
/=	División y asignación
%=	Calcular resto y asignación
^=	Elevar a y asignación

La anterior instrucción envía el flujo de datos a un fichero llamado *test*. Si el fichero no existe, AWK lo creará automáticamente. Se puede redirigir la salida utilizando > >. Puede utilizarse la redirección de la shell para este ejemplo tan sencillo, pero la variante AWK permite redirigir la salida a ficheros diferentes a la vez que lo imprime en pantalla.

AWK también entiende la función *printf()*, utilizada también en C y en la shell. Los administradores encontrarán útil el uso de *printf()* para formatos mejorados de salidas. Al igual que su equivalente en C, *printf()* no acaba en una nueva línea, sino que espera que el programador añada un *\n* para cada nueva línea.

```
gawk '{printf
("%x\n",$5)}' lista
```

En este ejemplo queremos que *printf* muestre un valor entero en formato hexadecimal (*%x*), para luego añadir un salto de línea (*\n*). El argumento que se le pasa es el contenido de la columna 5. Para más ejemplos, véase [2].

Un Nuevo Comienzo

Las estructuras *BEGIN* y *END* se utilizan para mostrar cabeceras o mensajes. AWK ejecuta cualquier instrucción *BEGIN* antes de analizar el fichero de entrada y cualquier instrucción *END* después de completar la última línea.

```
gawk 'BEGIN
{print "Buscar a MickeyMouse"}
$1=="MickeyMouse" {print $0}
END {print "-----"}' lista
```

Aparte de la manipulación de cadenas, AWK también puede vérselas con operaciones numéricas. La última línea en el fichero del Listado 1 contiene números que AWK puede manipular numérica o no numéricamente. Por ejemplo, se puede escribir la siguiente instrucción si se necesita saber cuánta memoria existe en tu entorno de laboratorio:

```
gawk '{sum+=$5; print $5,sum}'
lista
```

Este mini-programa suma el campo cinco de cada línea y almacena el total actual en una variable llamada *sum*. Muestra el valor del campo y de la suma.

Expresiones Regulares

Las expresiones regulares son útiles si se necesita manipular o buscar en documentos de texto. Los meta-caracteres proporcionan la posibilidad de crear pautas de búsqueda bastante complejas:

```
gawk '$1 ~ /[0-9]/ {print $0}'
lista
```

Este script busca en la columna uno una pauta de búsqueda entre 0 y 9. Para decirle a AWK que busque en *$1*, se necesita asignar explícitamente el número de columna a la pauta de búsqueda utilizando el carácter tilde (~). El operador de negación (*!~*) tendría el efecto opuesto. Con *$1 !~ /[0-9]/* buscaríamos las líneas donde no se encuentran las cifras entre 0 y 9.

Para buscar máquinas cuyos nombres acaban en *Duck*, escribiríamos la siguiente instrucción:

```
gawk '$1 ~ /Duck$/ {print $0}'
lista
```

El símbolo del dólar en *Duck$* le indica a gawk que busque a final del campo *$1*. Para encontrar entradas que comiencen con la palabra *Lucky*, utilizaríamos el símbolo circunflejo (^), así */^Lucky/* nos devolvería entradas como *LuckyLuke* o *LuckyDuck*. Los operadores booleanos ofrecen interesantes extensiones a estas funcionalidades:

```
gawk '$1 ~ /(y|M)/
{print $0}' lista
```

devuelve todos las líneas que contienen las letras *y* o *M* en el primer campo. La Tabla 2 muestra una selección de los meta-caracteres más comunes.

Funciones de Cadena

AWK dispone de una gran cantidad de funciones para la sustitución de cadenas. En nuestro fichero de ejemplo, sólo uno de nuestros ordenadores dispone de un sistema operativo Suse. Supongamos que el administrador que regenta esta red decide migrar a Debian y necesita actualizar la lista de ordenadores. En vez de utilizar un editor, el administrador podría realizar la tarea más elegantemente con AWK y la instrucción

```
gawk
'{sub(/Suse/, "Debian", $3);
print >> "/home/linux/test"}'
lista
```

Esta instrucción pasa la clave de búsqueda */Suse/*, el texto de sustitución "Debian" y la columna *$3* a la función *sub* ("substitution"). Suponiendo que la clave de búsqueda ocurre en esta columna, el texto es reemplazado. Si se precisa más información sobre funciones de cadena, recomendamos echar un vistazo a [3] y a la página man de gawk.

Cada Vez Más Grande

Los guiones AWK más complejos permiten definir tus propias funciones, bucles y matrices multidimensionales. La variante GNU incluso puede vérselas con comunicaciones TCP/IP [4]. ■

Tabla 2: Expresiones Regulares

Expresión	Significado
.	Coincide con cualquier carácter
^	Encuentra la expresión regular que le sigue al principio de una línea
$	Encuentra la expresión regular que le sigue al final de una línea
[]	Coincide con cualquier carácter entre los corchetes
[a-d1-7]	Coincide con los caracteres de un rango. En este caso: todas las letras de *a* hasta *d* y todos los dígitos de *1* hasta *7*
?	Coincide con 0 ó 1 de la expresión que le precede
*	Coincide con 0 ó más de la expresión que le precede
X|Z	O *X* o *Z*
XZ	*X* seguido de *Z*

RECURSOS

[1] Sitio de GNU AWK: *http://www.gnu.org/software/gawk*

[2] Ejemplos printf del manual de gawk: *http://www.gnu.org/software/gawk/manual/html_node/Printf-Examples.html*

[3] Helmut Herold, *AWK and SED* de Addison Wesley, 1991

[4] Comunicación TCP/IP con gawk: *http://www.gnu.org/software/gawk/manual/html_node/TCP_002fIP-Networking.html*

Consejos que nos ayudan a personalizar Bash. **POR BRUCE BYFIELD**

Una manera de reducir la ansiedad que genera el uso de la línea de comandos es conseguir todo el control sobre la situación que sea posible. Bash, la shell predeterminada en la mayoría de las distribuciones GNU/Linux, no es ninguna excepción. Si sabemos cómo personalizarla, comenzaremos a perder la sensación de angustia (sin duda inducida por una exposición al DOS que usábamos a comienzos de nuestra vida informática) que nos paraliza cuando se plantea la idea de usar la línea de comandos.

Evidentemente, muchas personalizaciones nos interesarán sólo si somos desarrolladores. Francamente, listar cada opción posible requeriría un artículo cinco o seis veces la longitud de éste. Aún así, los ejemplos que se muestran más adelante pueden interesar a usuarios de cualquier nivel y dan una idea de las posibilidades. Varían desde crear nombres cortos para comandos y cambiar permisos por defecto, a personalizar la apariencia y el sentido del prompt del comando y el comportamiento del historial de Bash.

Ficheros Implicados

Antes de comenzar hemos de saber que todas las cuentas de usuario poseen potencialmente cuatro ficheros asociados con Bash. Todos se encuentran normalmente ocultos, aunque podemos ver los que usa nuestra distribución escribiendo *ls -a .bash**.

Dos ficheros Bash son de interés limitado si estamos aprendiendo cómo personalizarla. El fichero *.bash_history* es una lista de comandos introducidos previamente, uno por línea. Aunque podemos editarla en un editor de texto, la mayoría de la gente utiliza las teclas Arriba y Abajo para desplazarse a través del historial para encontrar el comando que quieren volver a usar. El fichero opcional *.bash_logout* nos permite ejecutar un script cuando salimos de Bash, aunque no se utiliza a menudo.

Los otros dos ficheros Bash son fundamentales para la configuración. El primero de ellos, *.bashrc*, contiene configuraciones básicas para el historial y opciones del prompt, y está siempre presente. Si se borra, vuelve a crearse automáticamente por el fichero */etc/bash.bashrc*. El segundo es *.bash_profile*, el cual incluye opciones y configuraciones adicionales. Si no está presente, la cuenta del usuario usa */etc/profile* en su lugar, que es el predeterminado para el sistema completo.

Estos ficheros pueden editarse directamente o a través de un comando como *export*.

Cambiando la Ruta

La ruta es una lista de directorios que mira el sistema operativo en busca de comandos que introducimos. Consta de todas las rutas definidas en */etc/profile* más cualquiera añadida a *.bash_profile* en nuestro directorio de inicio. Si un comando no estuviera en la ruta, tendremos que introducir una ruta completa cuando escribimos el comando o bien cambiar al directorio en la que se encuentra – ninguna de las dos resultan tan convenientes como introducir el comando y confiar en Bash para saber dónde buscarlo.

Para añadir un directorio a la ruta, podemos abrir nuestro *.bash_profile* en un editor de texto y buscar el comando *$PATH*. Si no se menciona en ninguna parte del fichero, podemos introducir las líneas manualmente al final del mismo. Por ejemplo, si deseamos añadir un directorio */bin* para ejecutables a nuestro directorio de inicio, introduciríamos

```
PATH=$PATH:/home/mi_cuenta/bin
export PATH
```

Para que los cambios tengan efecto inmediato, hemos de ejecutar los comandos contenidos en el fichero con la instrucción *source*:

```
$ source /home/mi_cuenta/➥
.bash_profile
```

Si no, tendremos que esperar hasta la próxima vez que iniciemos sesión para apreciar los cambios.

Alternativamente, podríamos modificar la ruta desde la linea de comandos declarándolo primero y configurándolo después con:

```
$ PATH=$PATH:/usr/local/bin:➥
/usr/bin:/bin:/usr/games:➥
/home/mi_cuenta/bin
$ export PATH
```

Si decidimos que no necesitamos un directorio en nuestra ruta, podemos redefinirlo con las dos mismas líneas, omitiendo simplemente el directorio innecesario.

Nombres Cortos o Alias

Los comandos pueden llegar a ser largos y complejos. Dependiendo del comando, podemos tener el comando básico, cualquier número de variables, un fichero o directorio fuente y un objetivo. En algunos casos, como con el comando *apt-get*, también podemos tener un subcomando además del comando principal. Esta estructura puede ser difícil de recordar, y, para empeorar las cosas, un error en la sintaxis puede tener resultados no esperados o invalidar el comando.

De modo que Bash nos permite definir y usar atajos. Podremos crearlos editando *.bashrc* en un editor de texto, aunque es más rápido usar los comandos precompilados *alias* y *unalias*. Como indican sus nombres, *alias* crea atajos y *unalias* los borra.

Las estructuras de estos comandos son simples. Por ejemplo, si queremos ver siempre las listas de los contenidos del directorio codificadas en color, deberemos introducir el comando *alias ls = 'ls --color=auto'*.

Técnicamente, deberíamos comenzar con el *alias -p*, aunque la opción *-p*, que envía los resultados a una salida estándar, es innecesaria en todas las distribuciones que he probado, así que no necesitamos preocuparnos por ella.

Una vez que hemos definido este alias, en lugar de introducir siempre *ls --color = auto*, lo único que tenemos que hacer es escribir *ls*. Esto nos ahorra pulsaciones de tecla si usamos la línea de comandos para la administración de ficheros. Podemos hacer lo mismo para cualquier comando Bash o aplicación, incluyendo uno para el escritorio, si así lo decidimos. La limitación obvia es que hay que seleccionar atajos que sean poco probables de introducirse por accidente, aunque supongo que también puede que descubramos – al menos en teoría – algún alias que crea un conflicto entre opciones incompatibles.

Borrar un atajo es incluso más simple: Escribimos *unalias* seguido del nombre del atajo. Por ejemplo, si decidimos que utilizar colores en un listado de directorio no es algo que prefiramos después de todo (probablemente porque seamos daltónicos, o porque prefiramos la opción *-F* para indicar los tipos de ficheros por un carácter al final del nombre), entonces introducimos *unalias ls*.

Este comando borrará el alias, pero no – y permítanme que subraye esto – el comando *ls* en sí. Si deseamos borrar todos los alias, el comando es más simple aún: *unalias -a*.

Para ver una lista de atajos definidos, escribimos *alias* sin más. Si usamos muchos, deben añadirse líneas de comentarios (#) en *.bashrc* y organizar nuestros atajos por función para verlos luego en un editor de texto cuando necesitemos un recordatorio. En un fichero de muestra *.bashrc* que he visto recientemente, se habían separado los alias por categorías: programación, aplicaciones de escritorio, scripts y otra media docena de clases. Otros incluían errores de tecleo comunes, para que el usuario no recibiera un error al escribir *yum intsall* en vez de *yum install* cuando añade paquetes a su sistema Fedora. Tal y como muestran estos ejemplos, cómo de útil encontremos los alias dependerá completamente de nuestra paciencia e ingenio.

Tabla 1: Opciones del Prompt de Bash

\d	fecha
\h	nombre de la máquina
\t	hora actual en formato 24 horas HH:MM:SS
\T	hora actual en formato 12 horas HH:MM:SS
\@	hora actual en formato 12 horas am/pm
\A	hora actual en formato 24 horas HH:MM
\u	nombre de usuario del usuario actual
\w	directorio de trabajo actual, con el directorio de nivel superior indicado con una tilde (~)
\W	nombre base del directorio de trabajo actual, con el directorio de nivel superior indicado con una tilde (~)
* \\	barra invertida

Configurando Permisos Predeterminados

Los permisos definen cómo pueden usarse los ficheros o directorios. Cuando creamos un nuevo fichero, se está dando automáticamente un juego de permisos predeterminado denominado *umask*.

Una manera de resumir los permisos es utilizando tres dígitos. De izquierda a derecha, son los permisos de la cuenta de usuario desde la que se creó, para los que se encuentran en el mismo grupo que el creador y para todos los demás registrados. Aparentemente, sin más razón que la de que la idea pareció buena en su momento, cada dígito es la suma de números en base 8 que definen un permiso. El permiso de lectura es 4, el de escritura 2, el de ejecución 1, y ningún permiso es 0. A esta forma corta se le denomina modo absoluto, permisos simbólicos o permisos octales, según prefiramos.

Bajo este sistema, un fichero que todo el mundo pueda leer, escribir o ejecutar debería tener un permiso de *777*. Por el contrario, uno que el propietario podría leer, escribir y ejecutar y que nadie más podría usar, tendría un permiso de *700*.

Cuando creamos un fichero y el sistema asigna el juego predeterminado de permisos, el umask está definido en */etc/login.defs*. Sin embargo, podemos establecer el umask para cada cuenta añadiendo el comando *umask* seguido de los permisos (por ejemplo, *umask 700*) a *.bashrc*.

```
bruce@nanday:~$ ls -a .bash*
.bash_history  .bash_logout  .bash_profile  .bash_profile~  .bashrc
```

Figura 1: El comportamiento de Bash está configurado en cuatro ficheros principales en nuestro directorio de inicio.

Personalizando el Prompt

El prompt es el texto recurrente que aparece a la izquierda de donde comenzamos a escribir el comando. Si somos un usuario nuevo, posiblemente no le hayamos prestado demasiada atención.

Tampoco es absolutamente necesario en la actualidad. Antes de que los escritorios se convirtieran en la norma, el prompt contenía información útil que necesitábamos saber de un vistazo, como el directorio actual, la fecha y la hora. Hoy en día, que podemos conseguir toda esta información del escritorio, no deberíamos preocuparnos demasiado por él.

Aún así, incluso hoy en día, encontraremos útil disponer de información básica visible siempre, como la cuenta actual y el directorio, incluso a pesar de que estemos usando casi siempre un terminal virtual. Nunca sabemos cuándo necesitaremos reparar nuestro sistema y no tengamos disponible el escritorio. Además, si miramos algunas distribuciones, encontraremos a menudo que el carácter final en el prompt nos dice si estamos usando un usuario normal o una cuenta de root; los prompts de la cuenta del usuario acaban en un signo dólar, mientras que los de root lo hacen en un signo almohadilla (#).

De hecho, una vez que lo miramos, veremos que distribuciones diferentes poseen prompts diferentes basándose en qué información piensan que querrán ver los usuarios. Por ejemplo, si estuviera trabajando en el directorio */usr/share* en un ordenador llamado nanday, en Fedora, mi prompt predeterminado será *[bruce@conure share]$*, mientras que en Debian será *bruce@nanday:/usr/share$*. De esta diferencia deberíamos inferir que los valores predetermina-

```
bruce@nanday:~$ PS1="\u $ "
bruce $
```

Figura 2: Un cambio en el prompt de Bash dura solamente hasta que se cierra la sesión actual. Cualquier cambio en el prompt requiere referencias a una serie de símbolos arcanos.

dos de Fedora suponen que los usuarios permanecen gran parte del tiempo en sus directorios de inicio porque no se les da la ruta completa, mientras que los de Debian suponen que sus usuarios más avanzados pueden estar en cualquier parte en el sistema y prefieren no tener que usar el comando *pwd* para descubrir dónde se encuentran.

Además, posiblemente deseemos acortar el prompt, especialmente si estamos listando rutas a directorios anidados a cinco o seis niveles de profundidad. También puede que queramos cambiar el color del prompt para hacerlo más visible, para complacer nuestro sentido de la estética, o para resaltar mejor la diferencia entre la cuenta de root y todas las demás.

Sea cual sea nuestra razón, el mejor sitio para comenzar si queremos cambiar nuestro prompt es cambiar temporalmente el parámetro prompt *PS1*. Estos cambios permanecerán efectivos hasta que los cambiemos de nuevo o cerremos nuestra terminal virtual actual. El terminal siguiente que abramos nos volverá al prompt predeterminado.

Para comenzar, desearemos ver las configuraciones actuales para *PS1*, la variable para el prompt, introduciendo el comando *echo PS1*. Probablemente obtendremos una respuesta que dice algo como: *[\u@\h \W]$*, que es la lectura que consigo con una máquina con Fedora 10 instalado. Comparándolo con el prompt de Fedora mencionado antes, podemos entender lo que significa cada entrada. Nótense, sin embargo, los elementos extra, tales como la @ entre el nombre de usuario (*\u*) y el nombre del host (*\h*) y el espacio entre el nombre de host y el directorio actual (*\W*).

Podemos cambiar este prompt temporalmente tomando como referencia la Tabla 1. No me he molestado con algunas opciones que probablemente carezcan de interés para los modernos no-desarrolladores. Si deseamos consultarlas, podremos encontrar una lista completa online con una búsqueda rápida.

Por ejemplo, si quisiéramos presentar la cuenta de usuario actual, haríamos un cambio temporalmente ejecutando *PS1 = "\u $ "* para obtener el prompt *bruce $*. Nótese que el espacio después del signo dólar no es un error, sino que se añade deliberadamente para que lo que escribo esté separado del prompt.

```
bruce@nanday:~$ export PS1="\u $ "PS1="\e[0;31m[\u@\h \W]\$ \e[m "
[bruce@nanday ~]$
```

Figura 3: El prompt de Bash se cambia permanentemente, toma los efectos inmediatamente y los mantiene la próxima vez que iniciamos la shell.

Si deseamos añadir color, podemos cambiar el de los caracteres de acuerdo a la fórmula:

```
"\e[x;ym\e[m "
```

En ella, *\e[* marca el inicio de los caracteres a los que se les aplica el color, *\e[m* señala el final y *x;ym* es el código de color (Tabla 2). El comando para cambiar el prompt a rojo oscuro sería:

```
PS1="\e[o;31m[\u@\h \W]\$ \e[m "
```

Si sustituimos un *1* por un *0*, obtendremos una versión más suave del mismo color.

Podemos experimentar con estos cambios temporalmente hasta que consigamos el prompt que deseemos. Si cometemos un error, podemos usar las teclas de flecha para encontrar una entrada en el historial de comandos desde la que podemos cambiar el error o bien simplemente cerrar la ventana si estamos trabajando en un terminal virtual.

Cuando encontremos la fórmula del prompt que queremos, abrimos el fichero *.bashrc* para nuestra cuenta en un editor de texto y sustituimos la fórmula existente por la que hemos ideado. Si preferimos, podemos usar también el comando *export*: por ejemplo, *export PS1 = "\e[0;31m[\u@\h\W]\$ \e[m*.

Opciones del Historial

El historial Bash es una lista de comandos usados previamente almacenados en *.bash_history*. Si escribimos continuamente el mismo comando u otros similares, podemos usar las teclas de flecha Arriba y Abajo para desplazarnos hasta el comando que queremos usar.

El comando *export* configura las variables de la shell y es una manera de personalizar el historial Bash. O, editar directamente *.bashrc*. Para establecer el número de comandos almacenados en el historial a 1,200, bien lo introducimos en la línea de comandos o bien lo añadimos a *.bashrc*:

```
export HISTFILESIZE=1200
```

Como *.bash_history* es un fichero de texto, no requiere demasiada memoria, así que no hay razón alguna por la que no usemos esta configuración para incrementar el tamaño por defecto de 500 entradas. Evidentemente, cuanto mayor sea el fichero, más desplazamientos tendremos que hacer para encontrar un viejo comando, aunque siempre podemos abrir *bash_history* en un editor de texto y usar su función de búsqueda.

Otra personalización es mantener comandos duplicados que están añadidos a *.bash_history*. Para esta opción, introducimos:

```
export HISTCONTROL=erasedups
```

Esta personalización asegura que *.bash_history* contiene solamente comandos únicos, aunque podría significar que el comando que deseamos requiere desplazamiento extra, especialmente si hemos usado uno que ya aparece muy atrás en la lista.

Tabla 2: Códigos de Color

Negro	0;30
Azul	0;34
Verde	0;32
Turquesa	0;36
Rojo	0;31
Morado	0;35
Marrón	0;33

Sólo el Principio

Para llevar todo esto más lejos, necesitaremos saber cómo escribir scripts en Bash. Los scripts amplían las posibilidades radicalmente. En Debian, por ejemplo, un script se usa en *.bash_logout* para limpiar la pantalla para asegurar nuestra privacidad cuando salimos de la línea de comandos. Los ejemplos descritos en este artículo son lo suficientemente simples como para ayudarnos a comenzar y darnos una perspectiva de las posibilidades de personalización de Bash. ■

Configuramos el hardware desde la línea de comandos

HARDWARE ASISTIDO

Aprenderemos algunos comandos que nos permitirán descubrir y configurar nuestro hardware.

POR KLAUS KNOPPER

En los primeros días de Linux, los drivers internos del kernel eran los responsables de detectar y hacer funcionar los periféricos. Cualquier cambio de configuración hardware implicaba necesariamente la recompilación del driver. La llegada de los módulos del kernel hizo que el procedimiento de cambio de configuración del hardware fuera mucho más sencillo. Ya no era necesario ni recompilar ni reiniciar el sistema, siempre y cuando la parte principal del kernel funcionara correctamente.

Cuando los drivers no están siendo usados por ningún proceso, podemos eliminarlos de la memoria con el comando *rmmod*:

```
rmmod modulename
```

y pueden ser recargados con:

```
modprobe modulename ➥
option1=value1 option2=value2
```

Con el comando *modprobe* podemos pasar un conjunto de opciones que nos permitirán configurar cada módulo de manera independiente. Estas opciones son consultadas por el módulo en tiempo de ejecución. Para ver la lista de opciones válidas para un módulo en concreto, deberemos hacer uso del comando *modinfo*. Con

```
modinfo Image/KNOPPIX/➥
modules/cloop.ko
```

lograremos producir la salida del Listado 1. Los parámetros que nos muestra pueden ayudarnos a ajustar el módulo durante la fase de inicialización.

A partir de la versión 2.6 del kernel Linux, podremos cambiar muchos de los parámetros de los módulos sin necesidad de recargarlos. Para ello deberemos modificar los contenidos de los ficheros virtuales asociados al módulo, y localizados en */sys/module/modulename/parameters*.

Aparte de las opciones de configuración, *modinfo* nos muestra un conjunto de información asociada a la, aparentemente inocente, etiqueta *alias*. Suele aparecer varias veces y con valores diferentes. Esto lo podemos comprobar analizando la salida del comando *modinfo e1000*:

```
alias: pci:v00008086d0000➥
10B5sv*sd*bc*sc*i*
alias: pci:v00008086d0000➥
1099sv*sd*bc*sc*i*
```

Esta etiqueta nos muestra información hardware sobre los identificadores de producto y vendedor, aunque lo codifica en una secuencia formada por dígitos hexadecimales.

Por ejemplo, la segunda entrada revela que el módulo *e1000* debe responsabilizarse del hardware fabricado por Intel (con identificador de vendedor 8086) y cuyo identificador de dispositivo sea 1099.

Para poder cargar los drivers que coincidan con un determinado componente hardware, el sistema operativo necesita algún tipo de tabla que asocie el hardware actual con el correspondiente módulo que lo gestiona. El comando *lspci* proporciona un

mecanismo para listar el hardware disponible. Al escribir el comando sin parámetros

```
lspci
```

produce una pequeña lista de los dispositivos alojados en el sistema:

```
...
01:00.0 Ethernet controller: ➥
Atheros Communications Inc. ➥
AR5001 Wireless Network ➥
Adapter (rev 01)
03:00.0 Ethernet Controller: ➥
Attansic Technology Corp. L2 ➥
100 Mbit Ethernet Adapter ➥
(rev a0)
...
```

De manera alternativa, podemos escribir:

```
lspci -v -k
```

para la obtención de una salida con información más detallada (como la presentada en el Listado 2). Todo lo que muestra *lspci* se ha obtenido de la información que aloja el kernel y que almacena bajo los directorios virtuales */proc* y */sys*.

A partir del kernel 2.6, la mayoría de las distribuciones hacen uso del servicio *udev* para la carga automática de los módulos que casan con cierto hardware, y de HAL (capa de abstracción del hardware), que nos permite ejecutar scripts o programas una vez se detecta la presencia de hardware que requiere de la interacción con el usuario. Dependiendo de cómo se tenga configurado el escritorio podremos hacer que HAL, por ejemplo, nos muestre ventanas cada vez que introduzcamos un pendrive por el puerto USB.

Extrayendo Información de /proc y /sys

Los sistemas de ficheros *procfs* y *sysfs* ofrecen una funcionalidad similar. Con ambos sistemas de ficheros podremos conseguir información y establecer parámetros del sistema sin la necesidad de ninguna herramienta especial (con un simple *ls* podremos empezar a ver la información que proporcionan los directorios */proc* y */sys*).

Esa filosofía Unix en la que "todo es un fichero" es especialmente cierta dentro del directorio */dev* y sus ficheros asociados a dispositivos de entrada y salida. Esta filosofía se vuelve más evidente cuando analizamos el contenido de los ficheros que se encuentran bajo */proc* y */sys*.

El comando

```
cat /proc/cpuinfo
```

nos devuelve un pequeño resumen de la CPU de nuestro sistema (Listado 3).

El comando

```
cat /proc/interrupts
```

nos muestra cómo se reparten las interrupciones entre nuestras CPUs y los dispositivos del sistema. Este comando puede resultar útil a la hora de tener que diagnosticar problemas de conflicto hardware (Listado 4).

Si mostramos el contenido del fichero *version*

```
cat /proc/version
```

nos dirá bajo qué versión del kernel se está ejecutando nuestro sistema. Es el mismo resultado que obtendríamos al ejecutar *uname -a*.

Leyendo la información que se encuentra en */sys*, podremos programar pequeños y sencillos scripts de shell que nos listen los discos duros instalados en nuestro sistema (o cualquier otro dispositivo de bloque) y además nos muestren información hardware que se lea directamente desde el dispositivo. El script Bash mostrado en el Listado 5 recopila algunos fragmentos de información de los discos duros. Para ello lee */sys/block/diskname* y crea un menú desde el que poder seleccionar el dispositivo por su nombre (Figura 1).

Listado 1: Salida de modinfo

```
01 filename:
   Image/KNOPPIX/modules/cloop.ko
02 vermagic:       2.6.31.6 SMP pre-
   empt mod_unload modversions 386
03 description:    Transparently
   decompressing loopback block
   device
04 author:         Klaus Knopper (cur-
   rent maintainer), Paul Russel
   (initial Kernel 2.2 version)
05 license:        GPL
06 depends:
07 vermagic:       2.6.31.6 SMP pre-
   empt mod_unload modversions 386
08 parm:           file:Initial cloop
   image file (full path) for
   /dev/cloop (charp)
09 parm:           preload:Preload n
   blocks of cloop data into memory
   (uint)
```

Listado 2: Salida Detallada de lspci

```
01 ...
02 01:00.0 Ethernet controller: Ath-
   eros Communications Inc. AR5001
   Wireless Network Adapter (rev 01)
03 Subsystem: Device 1a3b:1026
04 Physical Slot: eeepc-wifi
05 Flags: bus master, fast devsel,
   latency 0, IRQ 18
06 Memory at fb3f0000 (64-bit,
   non-prefetchable) [size=64K]
07 Capabilities: [40] Power Manage-
   ment version 2
08 Capabilities: [50] MSI: Enable-
   Count=1/1 Maskable- 64bit-
09 Capabilities: [60] Express Legacy
   Endpoint, MSI 00
10 Capabilities: [90] MSI-X: Enable-
   Count=1 Masked-
11 Kernel driver in use: ath5k
```

Listado 3: /proc/cpuinfo

```
01 processor      : 0
02 vendor_id      : GenuineIntel
03 cpu family     : 6
04 model          : 13
05 model name     : Intel(R)
   Celeron(R) M processor 900MHz
06 stepping       : 8
07 cpu MHz        : 630.052
08 cache size     : 512 KB
09 fdiv_bug       : no
10 ...
```

Activando y Desactivando Dispositivos

El kernel de Linux ofrece cierta ayuda para la gestión de dispositivos y hardware especializado. Muchas veces resulta útil disponer de alguna aplicación gráfica que nos ayude a gestionar el hardware de forma sencilla.

Muchos portátiles disponen de teclas especiales (o combinaciones de teclas) que supuestamente deben activar o desactivar dispositivos como la WLAN o el bluetooth. Por desgracia, para que estas teclas puedan comunicarse con el hardware que controlan necesitan que se instale algún driver. Aunque esto es algo que depende del modelo y componentes del portátil, algunas veces es posible engañar al sistema y activar el dispositivo de forma manual – sin usar esas teclas especiales.

Un ejemplo que puede aplicarse a algunos dispositivos compatibles con Linux, es la función de permite activar y desactivar la WLAN que se encuentra en los Eee PC. A partir de la versión 2.6.30 del kernel, se incluye un driver especial que permite gestionar algunas de las características específicas del Eee PC. Nada más cargar el driver podremos gestionarlas desde el directorio */sys*:

```
$ find /sys -iname \*eeepc\*
/sys/bus/platform/devices/eeepc
```

Si miramos dentro del directorio *eeepc* que hemos conseguido encontrar gracias al comando *find*, veremos los ficheros de control que tenemos a nuestra disposición:

```
camera - activa o desactiva la cámara
cardr - activa o desactiva el lector de tarjetas SD integrado
cpufv - selector de frecuencia de la CPU y del Bus de datos
rfkill:rfkill0/state - interruptor de la WLAN
```

Para leer el estado actual del hardware sólo hay que ejecutar el siguiente comando:

```
cat /sys/bus/platform/devices/eeepc/camera
```

que responderá con *1* para "activado", y con *0* para "desactivado". Para establecer el dispositivo a un estado de "activo" podemos usar:

```
echo 1 > /sys/bus/platform/devices/eeepc/rfkill:rfkill0/state
```

que, en este ejemplo, activará la antena de la tarjeta WLAN integrada.

Así que ya sabe, cuando no encontremos la manera de activar la WLAN del portátil, una buena idea sería buscar algún fichero que tenga en el nombre la cadena *rfkill* dentro del directorio */sys*. Si lo encontramos, deberemos comprobar que podemos leer el estado actual, y cambiarlo escribiendo un valor *1* (o mayor). Es una prueba que merece la pena realizar antes de empezar a buscar nuevos drivers o programas para gestionar el interruptor *rfkill*. Por desgracia, en algunos portátiles este fichero es de solo lectura, al menos hasta que encontremos algún truco (o una nueva versión del driver) que evite este tipo de protecciones hardware.

En algunos portátiles, encender la luz de la WLAN (modificando el fichero llamado *wlan* o *wled* en algún lugar de */sys* o */proc*) activará la tarjeta WLAN además del propio led. El script del Listado 6 enciende la cámara, el lector de tarjetas y la WLAN, todo al mismo tiempo.

Podemos usar los eventos ACPI y las acciones gestionadas por este servicio (*acpid*) para asociar el pulsado de esas teclas especiales (o la combinación de ellas). Es posible configurar los eventos listados en */etc/acpi/event/** para llamar de forma individual a los scripts que realizan las acciones de activación del hardware.

Por ejemplo, */etc/acpi/events/power.conf* contiene las siguientes líneas:

```
event=button/power.*
action=/etc/acpi/actions/power.sh
```

Listado 4: /proc/interrupts

```
01 CPU0
02 0:   6870600  IO-APIC-edge
   timer
03 1:   211954  IO-APIC-edge
   i8042
04 9:   376971  IO-APIC-fasteoi
   acpi
05 12:  3639692  IO-APIC-edge
   i8042
06 14:      0  IO-APIC-edge
   ata_piix
07 15:   37391  IO-APIC-edge
   ata_piix
08 16:   444279  IO-APIC-fasteoi
   uhci_hcd:usb5, HDA Intel, i915@
   pci:0000:00:02.0
09 18:   4381110  IO-APIC-fasteoi
   uhci_hcd:usb4, ath
10 19:      0  IO-APIC-fasteoi
   uhci_hcd:usb3
11 21:      0  IO-APIC-fasteoi
   uhci_hcd:usb4
12 23:   748558  IO-APIC-fasteoi
   ehci_hcd:usb1, uhci_hcd:usb2
13 24:      1  PCI-MSI-edge   eth0
14 NMI:      0  Non-maskable inter-
   rupts
```

Listado 5: Información del Disco

```
01 for i in $(ls -1d /sys/block/?d?
   2>/dev/null); do
02 device="${i##/sys/block/}"
03 DEVICELIST[$((count++))]
   ="$device"
04 DEVICELIST[$((count++))]="$(cat
   $i/device/vendor 2>/dev/null)
   $(cat $i/device/model
   2>/dev/null) \
05 ($(awk '{print ($1 / 2048) "MB"}'
   $i/size 2>/dev/null))"
06 done
07 dialog --menu "Available
   Devices:" 10 50 7
   "${DEVICELIST[@]}" 2> choice.tmp
```

Listado 6: Encendiendo

```
01 for i in camera cardr
   rfkill:rfkill0/state; do
02 file=
   /sys/bus/platform/devices/eeepc
   /"$i"
03 [ -f "$file" ] && echo 1 > "$file"
04 done
```

Como vemos, la acción ejecuta el script */etc/acpi/actions/power.sh*, y desde ahí podremos iniciar la acción de apagar el ordenador, mostrar una ventana de confirmación o cualquier otra cosa que podamos pensar.

Ajuste del Sistema

Podemos optimizar la ejecución de Linux configurando de manera correcta los ficheros de */proc* y */sys*. Aunque siempre podremos hacer uso del comando *echo* para cambiar la configuración, existe una herramienta mejor: *sysctl*. Con ella podremos confeccionar configuraciones de sistemas persistentes (que se mantienen durante el reinicio del sistema), editando el fichero */etc/sysctl.conf*. Por ejemplo, el siguiente comando

```
echo 3000 > /proc/sys/vm/➥
dirty_writeback_centisecs
```

cambia el timeout del sistema, con lo que se guardarán en disco los cambios realizados cada 30 segundos.

Esta configuración es adecuada si deseamos acelerar el acceso de los portátiles a los discos de memoria sólida. Con ello lograremos aplicar el tiempo de vida útil (la frecuente escritura en disco puede ralentizar el sistema, y lo que es peor, nos acerca al número máximo de ciclos de escritura que soportan este tipo de memorias).

Esta configuración se mantendrá en nuestro sistema mientras no reiniciemos. Si queremos que sobreviva a las futuras cargas del sistema operativo, deberemos hacer esto mismo con *sysctl*:

```
sysctl -w vm.dirty_➥
writeback_centisecs=3000
```

Esto escribirá una línea en */etc/sysctl.conf* que podrá ser leída durante el reinicio del sistema:

```
vm.dirty_writeback_cen
tisecs=3000
```

Algunas veces es necesario establecer ciertos parámetros del kernel para activar alguna característica especial. El siguiente comando activa el reenvío de paquetes de red entre interfaces del red en IPv4, transformando el sistema en un router:

```
sysctl -w net-ipv4.ip_for
ward=1
```

(Este comando equivale a escribir un *1* en */proc/sys/net/ipv4/ip_forward*.) El reenvío IP del kernel quedará activado inmediatamente.

Conclusiones

Gracias a los sistemas de ficheros virtuales como */proc* y */sys*, conseguimos que el mecanismo de configuración del hardware se pueda realizar desde el espacio de usuario. Modificando ciertos ficheros en esos directorios, conseguiremos gestionar nuestros dispositivos desde la línea de comandos. Esto hace del shell una herramienta aun más potente para el control el sistema.

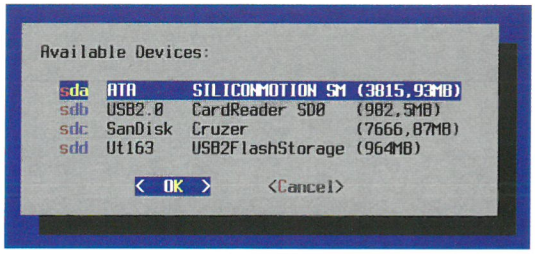


Figura 1: El script del Listado 5 nos muestra la información disponible para el dispositivo.

Ya no es necesario saber programar en C o ensamblador para reconfigurar nuestro hardware, a menos que, claro está, deseemos modificar el driver o escribir algún contolador nuevo para un hardware sin soporte. ■

Demonios y Scripts de Depuración

Los servicios del sistema, conocidos de forma tradicional como "demonios", se inician en las primeras etapas de la carga del sistema GNU/Linux. Muchos de estos servicios son realmente scripts. Esta es la razón por la que muchas distribuciones han decidido que los ficheros de configuración de esos servicios sean también pequeños scripts. De esta forma se evitan tener que inventar una nueva sintaxis para cada configuración. Estos ficheros de configuración son fácilmente reconocibles, ya que suelen tener:

- variables shell (escritas en mayúsculas y sin espacios) como BLUETOOTH_ENABLED=1 en */etc/default/bluetooth*,
- comentarios que explican el significado de cada variable, y
- (algunas veces) funciones shell que extienden o reescriben las ya existentes en otros scripts.

En los sistemas basados en Debian, la mayoría de estos scripts de configuración se encuentran en */etc/default/**, y el usuario *root* puede usar cualquier editor de textos para editarlos.

Si al final nos decidimos y editamos estos scripts, o si escribimos nuestros propios scripts de sistema, tendríamos que tener en cuenta ciertos detalles:

1. Si estos scripts son ejecutados como root, tienen el potencial de destruir el sistema si añadimos algún comando equivocado o, accidentalmente, activamos algún comando que se supone debería estar comentado.
2. Los scripts suelen llamarse desde otro script mediante el proceso de inclusión, usando el comando punto (.). En este caso hay que tener en cuenta que el script llamante terminará si se encuentra el comando *exit*.
3. No se permiten espacios antes y después del signo de asignación (=) a la hora de establecer valores a las variables. Los ficheros de configuración de KDE o Gnome contienen frecuentemente espacios por todos lados para facilitarnos su lectura, pero en el shell un espacio implica una separación entre el comando y sus parámetros, lo que puede desencadenar errores de sintaxis. Valores y opciones con espacios deben quedar entrecomillados.
4. Podemos comprobar la corrección sintáctica de un script sin necesidad de ejecutarlo. Para ello usaremos la opción *-n* de 'carga sin ejecución':

```
sh -n /etc/init.d/my-startup-script
```

Desafortunadamente, no todos los errores son detectados, ya que al no ejecutarse nada, ciertos parámetros incorrectos a llamadas de programas externos no se detectarán.

Si se desea depurar un script durante su ejecución, deberemos usar el comando *set*,

```
set -x
```

que imprime por pantalla cada comando antes de ser ejecutado, y

```
set +x
```

para finalizar la impresión de comandos.

Dos herramientas que saben de particiones de discos

PARTICIONES MÁGICAS

Creamos y examinamos particiones de discos Linux con fdisk y parted

POR NATHAN WILLIS

En sus inicios, Linux nos exigía que la partición de discos duros se hiciera manualmente. Hoy día los instaladores controlan esto automáticamente, incluso detectan otros sistemas operativos instalados y configuran correctamente una configuración multi-arranque. En la trastienda, sin embargo, la magia la realizan *fdisk* y *parted*. Conocer estas dos herramientas y el proceso de particionado es una ventaja para el administrador de sistemas.

Antes de comenzar, sería útil un rápido repaso a la terminología del particionado. Una partición es un "disco lógico" en un disco duro. Los lugares y tamaños de las particiones de discos se guardan en la tabla de particiones, escrita en el primer sector del disco. Las convenciones de la BIOS establecen que en un disco no puede haber más de cuatro particiones primarias. Por tanto, en la práctica, muchos usuarios crean una partición primaria de tipo "extendida", dentro de la cual pueden crearse más de cuatro particiones lógicas.

Cada partición puede contener exactamente un sistema de ficheros (la estructura de los datos que almacena ficheros y directorios), independientemente del tipo de sistema de ficheros de que se trate. El espacio swap está habitualmente implementado como un sistema de ficheros independiente y, consecuentemente, requiere su propia partición. Los discos extremadamente grandes se topan con otro problema de la BIOS: Ésta sólo puede leer los 1024 primeros cilindros del disco. Por eso, para asegurarse de que la imagen del kernel esté localizada dentro de ese límite, mucha gente crea una partición separada que contiene el sistema de ficheros */boot*. En consecuencia, a pesar de que Linux puede instalarse en un sistema de ficheros en una sola partición en teoría, probablemente encontraremos el sistema repartido en tres o más.

Históricamente, crear y gestionar particiones requería algún conocimiento de la geometría física del disco: el número de cilindros, pistas, cabeceras, etc. Las utilidades de particionado modernas, sin embargo, pueden conseguir toda la información que necesitan automáticamente, por lo que ya no es necesario.

Examinar Particiones

Fdisk puede utilizarse de dos maneras: en modo interactivo o no-interactivo. El segundo modo, o no-interactivo, únicamente solicita una partición e imprime la información de vuelta en *stdout*. El modo interactivo nos ofrece menús y opciones y nos permite modificar, además de explorar, particiones y tablas de particiones.

Al ejecutar *fdisk -l* se muestra un listado de la tabla de particiones para todos los discos del sistema (nota: para utilizar *fdisk* necesitaremos arrancarlo con *su* o *sudo*, dependiendo de nuestro sistema operativo). Para ver sólo una tabla de un disco único, lo añadimos al comando como parámetro (por ejemplo, *fdisk -l /dev/sda*). Nótese que *fdisk* requiere una *unidad* como argumento de dispositivo, el formato para ello es */dev/sdx* o */dev/hdx*. Las herramientas de sistemas de ficheros necesitan una *partición* como argumento de dispositivo, el cual consiste en el nombre del disco seguido del número de partición, como */dev/sda1* o *7dev/hdb4*.

La salida de fdisk (Figura 1) incluye el tamaño total del disco y la geometría básica, listando luego las particiones del disco y sus lugares de inicio y final, tamaño y tipo de partición (ambos por nombre y por número de ID). El tamaño se presenta en bloques. Añadiendo el parámetro *-u* podemos tener el informe fdisk de las posiciones de inicio y final de la partición en sectores en vez de en cilindros. Ejecutando *fdisk -s < /algún/dispositivo >* sólo se imprimirá el tamaño del dispositivo especificado, aunque funcionará tanto para discos como para particiones.

Parted también puede ejecutarse tanto en modo interactivo como no-interactivo. La sintaxis no interactiva es *parted < /algún/dispositivo > < comando >*. Ejecutando *parted /dev/sda* se imprimirá la tabla de particiones que se encuentra en el disco */dev/sda* (Figura 2). La información que se muestra es la misma que la de fdisk,

Gocke Goksel, Fotolia.com

excepto que parted no ofrece la opción de cambiar entre cilindros y sectores para la medida del tamaño.

fdisk

Para crear o cambiar particiones con fdisk comenzamos en modo interactivo: Omitimos los parámetros *-l* y *-s*, y especificamos un disco, como *fdisk /dev/hda* o *fdisk -u /dev/sdc*. El programa escribe su salida en la pantalla y proporciona un prompt de comandos, aunque no ofrece paginación (como *less*), así que tendremos que desplazarnos hacia arriba para leer las salidas largas. Introduciendo *m* en el prompt se listan los comandos disponibles de fdisk (Figura 3). Desde el menú principal podemos crear nuevas particiones (*n*), borrar las existentes (*d*), verificar la tabla de partición que hay (*v*) y configurar algunos parámetros (el más destacable es el parámetro *bootable*, combinado con *a*). Para aplicar cualquier cambio que realicemos, deberemos escribir una nueva tabla de partición al disco introduciendo *w*. En cualquier momento, con *q* podemos salir *sin* escribir la tabla de partición.

La creación de una partición es un proceso de múltiples pasos, aunque relativamente fácil. Empezamos introduciendo *n*, fdisk nos preguntará si deseamos crear una partición primaria (*p*) o una partición extendida (*e*). Indiferentemente de la opción que elijamos, fdisk nos pedirá a continuación que seleccionemos el número de partición (debemos ser cuidadosos de elegir una que no hayamos usado con anterioridad si ya hemos creado algunas particiones), el lugar del disco donde deseamos que comience la partición y su tamaño.

Fdisk sabe automáticamente el número del primer cilindro *disponible* en el disco, ya que conoce el tamaño de las particiones existentes, solicitándonos un valor como opción por defecto. Para dejar un espacio vacío entre particiones, escogemos un número más alto. Esto podría sernos de ayuda en caso de tener que redimensionar nuestra partición. De modo similar, fdisk nos pide un valor de tamaño por defecto, dándole el mayor posible. Sin embargo, no tenemos que aceptarlo y podemos especificar el tamaño de la partición que queremos en cilindros, sectores, megabytes o kilobytes.

Después de haber creado una partición, regresaremos al menú principal, donde podemos comprobar nuestra configuración con el comando *p*. Si es correcto, pulsamos *w* para escribir la nueva tabla al disco y salir.

```
magazine@MacBuntu: ~
File  Edit  View  Terminal  Help
magazine@MacBuntu:~$ sudo fdisk -l

WARNING: GPT (GUID Partition Table) detected on '/dev/sda'! The util fdisk doesn't support
  GPT. Use GNU Parted.


Disk /dev/sda: 500.1 GB, 500107862016 bytes
255 heads, 63 sectors/track, 60801 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1               1          26      204819+  ee  GPT
/dev/sda2   *          26       47773   383528984   af  HFS / HFS+
/dev/sda3           47774       47907     1068608   83  Linux
/dev/sda4           54786       60779    48146805   83  Linux
magazine@MacBuntu:~$
```

Figura 1: Listamos las tablas de las particiones con fdisk.

Introduciendo *x* desde el menú principal nos lleva hasta el menú de "funcionalidad extra", desde el cual podemos realizar tareas menos comunes y más arriesgadas, como cambiar el número de cilindros o cabeceras (Figura 4). La tarea más avanzada que probablemente podamos llevar a cabo es fijar el orden de partición (*f*). Al hacerlo, se cambian los *números* de partición en la tabla de particiones de modo que sean secuenciales. Los números pueden desordenarse si creamos y borramos particiones frecuentemente. Esta función lo corregirá, aunque debemos *ajustar* las referencias a las particiones en ficheros de sistemas como */etc/fstab*.

parted

El modo interactivo de parted se inicia con el parámetro *-i* antes de los argumentos del comando y del dispositivo. Sin embargo, a diferencia de fdisk, también podemos crear y modificar particiones en modo no interactivo. Parted soporta algunos comandos relativos al mantenimiento del sistema de ficheros y no para las particiones, aunque esto simplifica algunos escenarios comunes, tales como la creación de una nueva partición y crear un sistema de ficheros en ella con un único comando. Pero parted es útil más allá de la creación simultánea de una partición y de un sistema de ficheros: Si preferimos dejar las tareas relacionadas con el sistema de ficheros para otro momento y el sistema de ficheros que deseamos crear no es una de las opciones soportadas por parted.

Otra diferencia importante entre parted y fdisk es que los comandos de parted tienen efecto inmediato: Es decir, cuando creamos una nueva tabla de partición, la que ya existe en el disco (si la hubiera) se sobrescribe. Esto deja poco margen de error cuando trabajamos con un disco en el que ya existen particiones: un paso en falso podría borrar la partición errónea y dejarnos buscando herramientas de recuperación (afortunadamente, parted también puede ayudarnos en ese tipo de tareas).

El comando *parted < /algún/dispositivo > mklabel < tipo >* crea una nueva tabla de partición en el dispositivo especificado.

```
magazine@MacBuntu: ~
File  Edit  View  Terminal  Help
magazine@MacBuntu:~$ sudo parted /dev/sda print
[sudo] password for magazine:
Model: ATA Hitachi HTS54505 (scsi)
Disk /dev/sda: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start   End     Size    File system     Name                  Flags
 1      20.5kB  210MB   210MB   fat32           EFI System Partition  boot
 2      210MB   393GB   393GB   hfs+            Untitled
 3      393GB   394GB   1094MB  ext3
 6      394GB   394GB   1000kB                                        bios_grub
 7      394GB   448GB   54.2GB  ext4
 8      448GB   451GB   2358MB  linux-swap(new)
 4      451GB   500GB   49.3GB  ext3
 5      500GB   500GB   181MB   linux-swap(new)

magazine@MacBuntu:~$
```

Figura 2: Listamos las tablas de las particiones con parted.

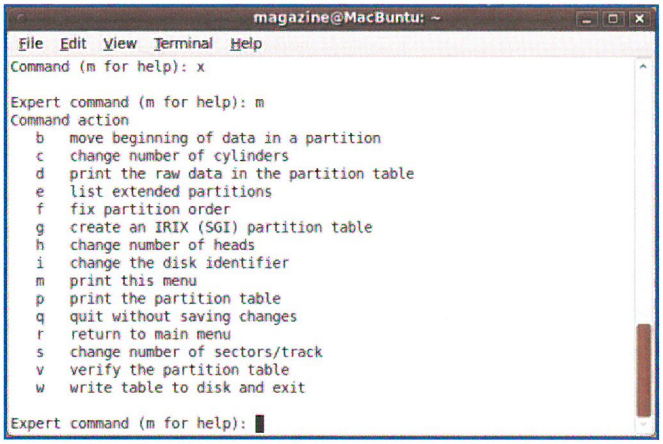


Figura 3: Comandos de fdisk.

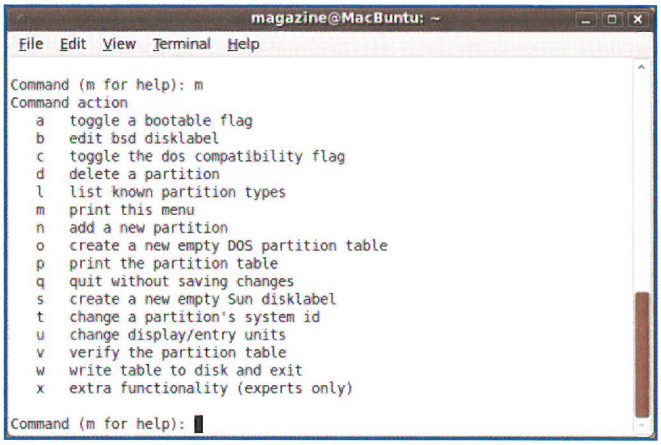


Figura 4: Comandos expertos de fdisk.

Con este comando debemos especificar el tipo de tabla de partición. En Linux, uno extensamente utilizado es *msdos*, aunque parted soporta algunos otros, incluyendo unos cuantos de otros sistemas operativos como Unix. Para crear una nueva partición utilizamos:

```
parted < /algún/dispositivo > ↵
mkpart < tipoPartición > ↵
< tipoSistemaFicheros > ↵
< inicio > < fin >
```

donde *tipoPartición* es primaria, extendida o lógica. Para particiones primarias o lógicas también deberemos especificar *tipoSistemaFicheros*, el formato de sistema de ficheros que tendrá la partición. El comando *mkpart* no crea de hecho el sistema de ficheros. Si queremos que lo haga, utilizaremos en su lugar *mkpartfs*. Los formatos de sistemas de ficheros soportados son *ext2*, *FAT32*, *FAT16*, *HFS*, *NTFS*, *ReiserFS*, *UFS* y *Linux swap*. Los parámetros de *inicio* y *fin* especifican el lugar de la nueva partición en el disco. No necesitamos utilizar geometría de discos como sectores, aunque podemos ofrecer tamaños expresados en términos humanos en megabytes.

Para eliminar una partición usamos

```
parted ↵
< /algún/dispositivo > rm <N>
```

donde *N* es el número de la partición. Por ejemplo, *parted /dev/hda rm 5* borra la partición */dev/hda5*. Parted también puede cambiar un número de parámetros comunes con el comando *set*. *parted /dev/sdc set 3 boot on* configura el parámetro de arranque en */dev/sdc3*.

Donde realmente difiere parted de fdisk es en su capacidad para mover y cambiar el tamaño de las particiones. Con objeto de que se vea la variedad, se aconseja iniciar parted en modo interactivo antes de explorar la manipulación de la partición y seguiremos necesitando un argumento de dispositivo del disco en modo interactivo, como puede ser *parted -i /dev/hdb*. Aunque en modo interactivo se supone el dispositivo dado como un argumento, no necesitamos incluirlo en los comandos que introducimos. Para cambiar a un dispositivo diferente dentro del modo interactivo, escribimos *select < /algún/otro/dispositivo >* en el prompt del comando parted.

El comando *resize* toma tres argumentos: el número de la partición, el nuevo lugar de inicio de la partición que ha cambiado de tamaño y el nuevo lugar del final. Para continuar el ejemplo de arriba, ejecutamos *resize 1 0 1000* en el prompt y nos cambiará el tamaño de la partición en */dev/hdb1* para comenzar en el inicio del disco y acabar en la marca de los 1000MB. Parted puede agrandar y reducir particiones, pero debe haber suficiente espacio libre en disco en el sistema de ficheros para acomodar la reducción y el suficiente espacio no asignado en el disco para acomodar la expansión. Nótese, sin embargo, que para los sistemas de ficheros ext2 y ext3, no podemos cambiar el lugar de inicio con un comando *resize*, solamente el final. Esta restricción no se aplica a otros sistemas de ficheros que soportan parted. Parted puede mover una partición a un nuevo sitio libre en el disco con el comando *move*. La sintaxis es *move < númeroPartición > < inicio > < fin >*, aunque *fin* es opcional. Si se omite, la partición se cambiará al nuevo lugar con su tamaño original. Si se da un nuevo punto final de tal modo que implica un nuevo tamaño para la partición, parted modificará el tamaño automáticamente y moverá la partición.

Cuando se mezclan y se cambian los tamaños de las particiones, a veces necesitaremos duplicar una partición en un nuevo sitio, quizás mover una partición a un nuevo dispositivo en un intento de liberar más espacio. En el prompt de parted utilizamos:

```
cp ↵
< /dispositivo/original > ↵
< particiónOriginal > ↵
<particiónObjetivo >
```

donde */dispositivo/original* es opcional – si se omite, se supondrá el dispositivo de trabajo actual. Por tanto, el comando *cp /dev/sdb 5 1* copiará la partición */dev/sdb5* a */dev/sdb1*. Por otro lado, *cp 5 1* copiará */dev/hdb5* a */dev/hdb1*.

Si mover y cambiar el tamaño de las particiones no fuera ya suficiente, parted tiene aún otra misión: rescatar una partición. Si, por ejemplo, borramos accidentalmente una partición de la tabla de particiones o sobrescribimos la tabla en sí, *rescue < inicio > < fin >* iniciará una búsqueda para firmas del sistema de ficheros en el disco. Parted buscará en un rango de sectores en torno a las posiciones de inicio y final buscando signos del sistema de ficheros, por lo que no necesitamos ser exactos. Si encuentra lo que cree que es un sistema de ficheros en el lugar apropiado, parted nos preguntará si deseamos crear una nueva partición para él. Para que funcione este rescate, sin embargo, el sistema de ficheros necesita estar más o menos intacto: parted sólo es capaz de recrear las entradas de la tabla de partición para restaurar la partición – para resolver la corrupción del sistema de ficheros necesitaremos otras herramientas. ■

Configuración de sistemas de ficheros con mkfs, df, du y fsck.

CONSTRUCTOR

A pesar de que la mayoría de las distribuciones Linux actuales cuentan con interfaces gráficas de uso simple para configurar y gestionar sistemas de ficheros, conocer cómo llevan a cabo estas tareas es útil. Mostramos cómo configurar y gestionar sistemas de ficheros con *mkfs*, *df*, *du* y *fsck*. **POR NATHAN WILLIS**

Stanislaw Kornogorov, Fotolia.com

Linux soporta un extenso abanico de tipos de sistemas de ficheros, incluyendo muchos que se originaron en otros sistemas operativos. Las opciones más comunes para discos duros, sin embargo, siguen siendo los ext2 y ext3 (el sucesor de ext2) nativos de Linux, seguido por los sistemas de ficheros de alto rendimiento XFS y ReiserFS. Por compatibilidad, también es importante saber cómo trabajar con el sistema de ficheros VFAT, ya que es el estándar que se encuentra preinstalado en muchos medios, incluyendo pendrives USB y discos flash.

Finalmente, algunas de las mismas utilidades que se utilizan para gestionar sistemas de ficheros normales también se aplican a particiones swap, las cuales son usadas por el kernel de Linux como memoria virtual cuando la RAM escasea.

mkfs

El comando *mkfs* (Figura 1) crea un nuevo sistema de ficheros en un dispositivo de bloque especificado, como una partición en un disco duro. El uso básico es

```
mkfs -t ➥
tipoSistemaFicheros ➥
/algún/dispositivo
```

donde *tipoSistemaFicheros* es un tipo de sistema de ficheros soportado por Linux (por ejemplo, ext2 o xfs) y */algún/dispositivo* es el lugar de la partición del disco objetivo (por ejemplo, */dev/hda1* o */dev/sdc3*). Las opciones específicas del sistema de ficheros se añaden después de *tipoSistemaFicheros*.

mkfs encarga la creación del sistema de ficheros a una de las múltiples utilidades especializadas, dependiendo del tipo de sistema de ficheros que especificamos: *mkfs.ext2*, *mkfs.xfs* o *mkfs.vfat*, por ejemplo. Como los sistemas de ficheros difieren tanto unos de otros, disponer de herramientas especializadas mantenidas por expertos en los sistemas de ficheros individuales da como resultado un código más estable.

La mayoría de dichas utilidades implementan las mismas opciones, aunque varían según las funcionalidades implementadas en los distintos sistemas de ficheros. A pesar de que *mkfs* invoca a esas otras utilidades, nótese que generalmente deberemos ejecutar el comando estándar *mkfs* cuando creamos un sistema de ficheros en vez de ejecutar cualquiera de las utilidades especializadas directamente.

A pesar de las diferencias, existen unas cuantas opciones que son comunes a todas las utilidades *mkfs.**. Añadiendo el parámetro *-c* verificará el dispositivo especificado en busca de bloques malos, los cuales se omitirán durante el paso de creación del sistema de ficheros actual. Añadir los parámetros *-v* o *-V* produce salidas detalladas o extremadamente detalladas, respectivamente.

Ejemplos de mkfs

Para formatear la primera partición del primer disco ATA de la serie con un sistema como ext3, ejecutaríamos el comando:

```
mkfs -t ext3 /dev/sda1
```

Esta instrucción utiliza el tamaño de bloque, parámetros inodo y todas las otras opciones predeterminadas, algunas de las cuales de hecho se determinan en tiempo de ejecución cuando mkfs analiza la geometría de la partición del disco.

El siguiente comando

```
mkfs -t ext3 -b 4096 /dev/sda1
```

también crea un sistema de ficheros ext3 en */dev/sda1*, aunque forzará el

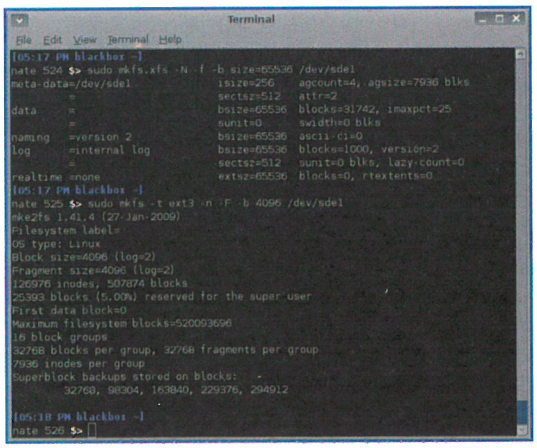


Figura 1: Los comandos mkfs simulados para XFS y ext3 difieren (los parámetros -N y -n especifican una simulación, la cual, de hecho, no crea un sistema de ficheros). Los parámetros -f y -F le dicen a mkfs que fuerce la creación del sistema de ficheros, incluso si ya detecta uno en el mismo sitio.

uso de bloques de 4.096 bytes. Ejecutando *mkfs -t ext3 -b 4096 -J device=/dev/sdb1 /dev/sda1* creará el mismo sistema de ficheros que el comando anterior, aunque en el caso de la bitácora, lo hrá en una partición separada, concretamente en */dev/sdb1*.

Para crear una partición XFS en */dev/sda1*, introducimos el siguiente comando:

```
mkfs -t xfs /dev/sda1
```

Para especificar el uso de bloques de 4.096 bytes en este sistema de ficheros, utilizamos *mkfs -t xfs -b size=4096 /dev/sda1*, una sintaxis diferente a la utilizada por ext3. De modo similar, *mkfs -t reiserfs /dev/sda1* crea un sistema de ficheros ReiserFS con las configuraciones por defecto. Para cambiar el lugar de la bitácora a */dev/sdb1* ejecutaríamos:

```
mkfs -t reiserfs -j ➥
 /dev/sdb1 /dev/sda1
```

Las variaciones en la sintaxis hacen especialmente crítica la referencia a la página man para más de un uso de *mkfs* con opciones de sistemas de ficheros específicas.

Mantenimiento de Rutina

A pesar de que el tamaño de los discos duros se incrementa cada año, parece que cada vez tardan menos en llenarse. El que se agote el espacio en un sistema de ficheros es uno de los problemas más comunes con los que probablemente nos encontremos en un sistema Linux, y no se trata sólo de un inconveniente debido a razones de almacenamiento – el uso de sistemas de ficheros temporales significa que un completo o casi completo sistema de ficheros raíz puede interferir con operaciones normales.

Para comprobar el uso de sistemas de ficheros empleamos *df* (Figura 2). Cuando no se añaden argumentos, *df* devolverá una tabla resumiendo el uso de los sistemas de ficheros montados, en kilobytes y con un porcentaje del espacio llenado del total de cada sistema de ficheros. Para obtener un informe para un sistema de ficheros en particular, lo especificamos como argumento, como por ejemplo, *df /dev/sda1*. También, podemos pasar un nombre de fichero como argumento, y *df* informará de los sistemas de ficheros que contiene el fichero especificado – lo cual podría resultarnos útil si no recordamos dónde se encuentra montado un sistema de ficheros determinado. Finalmente, existen unas cuantas opciones que hacen que *df* sea de mayor utilidad: *-i* informa del uso de inodos en vez del uso de bloques del sistema de ficheros; *-l* limita el informe a sistemas de ficheros locales solamente; *--type=tipoSistemaFicheros* y *--exclude=tipoSistemaFicheros* nos permiten limitar o excluir de la comprobación ciertos tipos de sistemas de ficheros, respectivamente.

Al descubrir un sistema de ficheros casi completo, podemos explorar el uso del espacio con *du*. Ejecutando *du /un/directorio* obtendremos una lista del espacio ocupado en el disco por cada subdirectorio bajo */un/directorio*, expresado en kilobytes. Añadiéndole la opción *-a*, *du* nos dará el espacio utilizado por los ficheros además de los directorios.

Ambos comandos son recursivos. Si no proporcionamos a *du* un directorio como argumento, nos informará del directorio actual. La opción *-c* presenta un total global además de las estadísticas de uso individuales. Hay otra opción que podría ayudarnos a localizar un fichero grande: *-L*, que sigue todos los enlaces simbólicos; *-x*, que limita el alcance de la búsqueda al sistema de ficheros actual solamente; y *--max-depth=N*, que nos permite limitar el número de subdirectorios recursivos por los que bajamos. Esta opción es muy útil cuando tratamos con una librería de ficheros grande.

Algunas utilidades existen para ayudarnos a conseguir un mejor rendimiento de nuestros sistemas de ficheros. El programa *tune2fs* nos permite controlar los parámetros de los sistemas de ficheros ext2, ext3 y ext4; podemos configurar el número de montajes entre los controles automáticos de integridad del sistema de ficheros con *tune2fs -c N*, estableciendo el intervalo de tiempo máximo entre controles con *tune2fs -i N[d\|m\|w]*, donde *d*, *m* y *w* son días, meses y semanas, respectivamente; o añadir una bitácora ext3 a un sistema de ficheros que no tiene ninguna con *tune2fs -j*. También podemos adaptar los parámetros RAID, las configuraciones de bitácora y el comportamiento de los bloques reservados, además de cambiar los parámetros manualmente, como el tiempo desde la última comprobación y el número de montajes, de los cuales se informa normalmente de manera automática.

ReiserFS posee una utilidad *tunefs.reiserfs* que sigue la misma premisa, aunque de momento sus opciones se limitan en su mayor parte a cambiar las opciones de bitácora, incluyendo el tamaño de la bitácora, el lugar y el tamaño de transacción.

ReiserFS posee utilidades separadas para modificar el tamaño de los sistemas de ficheros (*resizefs.reiserfs*) y copiar un sistema de ficheros desde un dispositivo a otro (*cpfs.reiserfs*). La herramienta *resize2fs* puede cambiar el tamaño tanto de ext2 como de ext3. Todas las herramientas dedicadas a modificar el tamaño pueden ampliar o reducir un sistema de ficheros, aunque ampliarlo requiere que la partición del disco subyacente tenga suficiente espacio vacío – el sistema de ficheros no puede crecer más allá del límite de la partición.

XFS posee su propia serie de utilidades que cubren la mayor parte de las

mismas opciones. El comando *xfs_growfs* puede cambiar el tamaño de un sistema de ficheros XFS, aunque puede hacer otros cambios también. Usando la opción *-m* con *xfs_growfs*, podemos cambiar la cantidad de espacio reservado para los inodos, y las opciones *-l* y *-L* nos permiten hacer ajustes a la bitácora – ambas funcionalidades se encuentran en otras utilidades en los otros sistemas de ficheros.

XFS también ofrece una herramienta de defragmentación llamada *xfs_fsr* que puede defragmentar un sistema de ficheros XFS montado; no existe una utilidad como ésta para ext2, ext3 o para ReiserFS. Adicionalmente, podemos realizar copias de seguridad y capturas del sistema de ficheros de XFS. La herramienta *xfs_freeze* inmoviliza la E/S en un sistema de ficheros. *xfsdump* escribe una copia de seguridad de un sistema de ficheros (en orden de inodo, por lo cual puede ejecutarse en un sistema de ficheros montado) y *xfsrestore* lo restaura de una copia previa.

Resolución de Problemas

Linux chequea cada sistema de ficheros periódicamente durante el arranque, buscando inconsistencias tales como inodos que no parecen estar asociados con un fichero, desajustes entre el número de enlaces del inodo y la cuenta de enlace del inodo, o desajustes entre el número de bloques libres en el sistema de ficheros y el número esperado, tal y como está registrado en el superbloque. Todos estos problemas son indicativos de que podría haber ocurrido un problema o un cuelgue.

La utilidad que lleva a cabo la comprobación es *fsck*. Si sospechamos de la existencia de problemas en un sistema de ficheros, podemos ejecutar

```
fsck /un/dispositivo
```

para realizar una comprobación manual y hacer las reparaciones necesarias. Si ejecutamos *fsck* sin un dispositivo especificado, ejecutará comprobaciones secuencialmente en todos los sistemas de ficheros en */etc/fstab*.

Los programas de comprobación de errores específicos de sistemas de ficheros – *e2fsck* para ext2 y ext3, *reiserfsck* para ReiserFS, y *fsck.vfat* para VFAT (Figura 3) – soportan muchas de las mismas opciones, pero de nuevo, la sintaxis puede variar, de manera que es crucial leer la página man para el corrector del sistema de ficheros antes de intentar ninguna reparación.

Cuando está corrupto, el sistema de ficheros VFAT sufre un conjunto de problemas diferentes: clusters malos, indicadores de directorios malos, incluso nombres de ficheros erróneos. La herramienta *fsck.vfat* puede encontrar y corregir muchos de estos problemas. Al igual que las herramientas mencionadas, puede invocarse en modo no-interactivo para uso en scripts, y puede marcar clusters malos automáticamente para evitar su reutilización en el futuro. El parámetro *-V* le dice a *fsck.vfat* que ejecute una segunda comprobación después de que ha intentado corregir todos los errores.

XFS ha separado las utilidades de comprobación de errores y la de reparación en *xfs_check* y *xfs_repair* respectivamente. Al igual que con los otros sistemas de ficheros con bitácora, existe una opción para apuntar a la bitácora en un dispositivo externo. Dos funcionalidades útiles exclusivas de *xfs_check* son el parámetro *-f*, el cual realiza la comprobación en una imagen del sistema de ficheros almacenado como un fichero normal (como la copia de seguridad de un sistema de ficheros creado con *xfs-dump*), y el parámetro *-s*, el cual especifica que sólo se informa de los errores más serios. La herramienta *xfs_repair* puede corregir la mayoría de los mismos problemas de corrupción tratados por las utilidades *e2fsck*, *reiserfsck* y los otros *fsck.**.

Un examen más flexible de un sistema de ficheros problemático requiere otros programas. Para problemas de ext2 y ext3, la herramienta *debugfs* nos permite examinar un sistema de ficheros y corregir errores interactivamente. *debugfs* puede realizar operaciones paso a paso y trabajar dentro de un sistema de ficheros con comandos similares a los de una shell Linux típica, tales como *cd*, *open*, *close*, *pwd*, *mkdir* e incluso *chroot*. Su poder real le llega a través de su capacidad para examinar superbloques, bloques e inodos directamente, repartiéndolos o asignándolos y no asignándolos individualmente, liberando bloques e incluso creando enlaces. ■

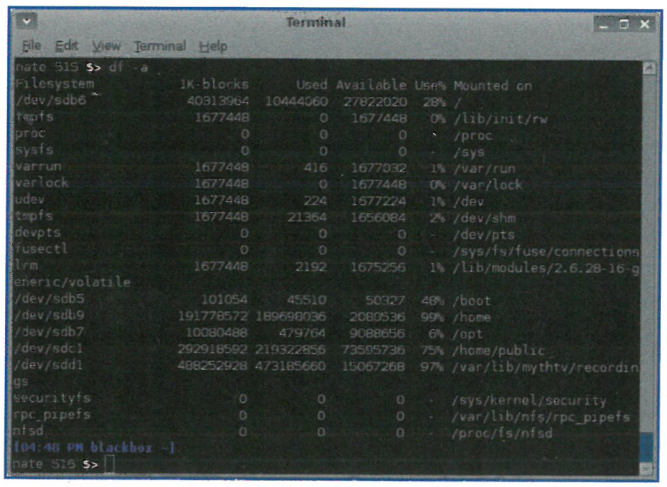


Figura 2: Los resultados de un comando df muestran en directo el uso del disco en un sistema. El parámetro -a incluye sistemas de ficheros "falsos" como /proc en la salida.

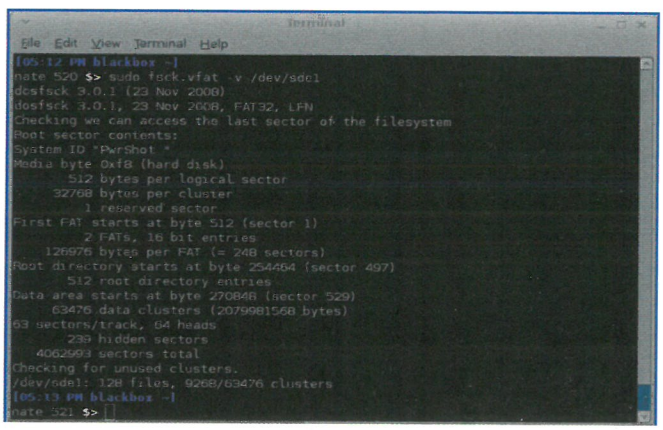


Figura 3: Los resultados de fsck con las opciones por defecto en un sistema de ficheros VFAT. El parámetro -v nos da una salida prolija. Los sistemas de ficheros con bitácora (ext3 y ReiserFS) poseen opciones que nos permiten especificar el lugar de la bitácora si se almacena en un dispositivo diferente.

shyle-zacharias; sxc.hu

Cal, Date, Hwclock y NTP

ATRAPADO EN EL TIEMPO

No por mucho madrugar amanece más temprano. Sin embargo, para muchas aplicaciones es importante que el ordenador tenga la fecha y hora correctas. Vemos cómo mantener el tic-tac del reloj de nuestro ordenador en marcha y cómo usar NTP para sincronizar la hora con un servidor de hora en la web.

POR HEIKE JURZIK

Un reloj de ordenador configurado incorrectamente puede ser desastroso: si nuestro ordenador pierde la pista del tiempo, podríamos acabar haciendo malabares con ficheros del futuro o correos de hace 30 años. El tiempo detenido podría llevarnos a malentendidos, errores e incluso a que el sistema se cayera.

Casi todas las distribuciones Linux configuran la hora y la zona horaria durante la instalación, y entornos de escritorio como KDE o Gnome muestran un reloj en el panel para permitir a los usuarios un acceso rápido a herramientas que posibilitan la configuración del reloj del ordenador (Figura 1).

En la shell, *cal* presenta un simple calendario ordenadamente formateado. El comando *date* nos da la fecha y la hora, aunque la salida por sí misma es bastante escasa. Además, este programa puede ayudar al administrador a configurar la fecha y la hora. *date* también demuestra su potencial en combinación con otras herramientas de la línea de comandos y en scripts. Por ejemplo, cuando el programa genera nombres de ficheros que contienen la fecha actual.

La herramienta *hwclock* ayuda a sincronizar el sistema horario y el reloj del hardware. Evidentemente, necesitaremos ser root para ejecutar estos programas.

Si el PC posee una conexión a Internet permanente, podemos automatizar el proceso de configuración del reloj sincronizando nuestro propio cronometrador horario con un servidor en la web a través del protocolo Network Time Protocol (NTP).

Calendario de la Línea de Comandos

Si llamamos al calendario, *cal*, sin ningún parámetro adicional, el programa presentará el mes actual del año en curso. La opción *-3* le dice que muestre el mes siguiente y el pasado; el parámetro *-y* ofrece un calendario para el año completo (Figura 2).

Para que cal nos presente un mes específico deberemos pasárselo bajo la forma numérica de dos dígitos, mientras que para el año utilizamos un número de cuatro dígitos. Por defecto, cal nos mostrará el calendario en la lengua definida en la variable de entorno *LANG* [1]. Si preferimos el formato de fecha y hora en cualquier otro

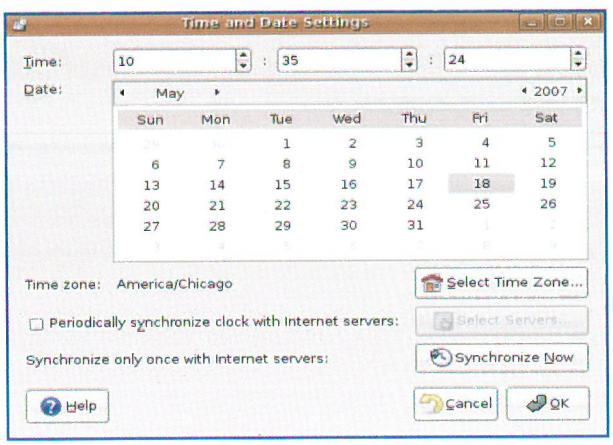


Figura 1: Los usuarios de KDE y Gnome pueden acceder a las configuraciones de fecha y hora con un clic sobre el reloj del panel.

lenguaje, aunque nos gustaría mantener la salida de todos los demás programas en el lenguaje predeterminado, podemos configurar la variable *LC_TIME* para decirle que use el lenguaje elegido.

El ejemplo siguiente establece la fecha y hora en Inglés:

```
LC_TIME=C cal -y
```

Es obvio que este comando no es necesario si la lengua por defecto ya está establecida como Inglés.

Para poner el calendario en perfecto castellano:

```
LC_TIME="es_ES" cal -y
```

Hacemos las variables permanentes añadiendo el comando *export* al fichero *~/.bashrc*, como en *export LC_TIME="es_ES"*.

¿Qué Hora es?

Si escribimos *date* en la línea de comandos veremos la fecha, la hora y la zona horaria:

Tabla 1: Fecha: Parámetros de la Línea de Comandos

Parámetro	Significado
%M	Minutos (00 a 59)
%H	Horas, reloj de 24 horas
%I	Horas, reloj de 12 horas
%a	Día de la semana, forma corta
%A	Día de la semana, forma larga
%d	Día como un número de dos dígitos
%b	Nombre del mes, forma corta
%B	Nombre del mes, forma larga
%m	Nombre como número de dos dígitos
%y	Año como número de dos dígitos
%Y	Año como número de cuatro dígitos
%D	Fecha de cuatro dígitos (mm/dd/yy)
%T	Hora en formato reloj 24 horas (hh:mm:ss)
%r	Hora en formato reloj 12 horas (hh:mm:ss)
%t	Tabulador
%n	Nueva línea
%%	Signo de %

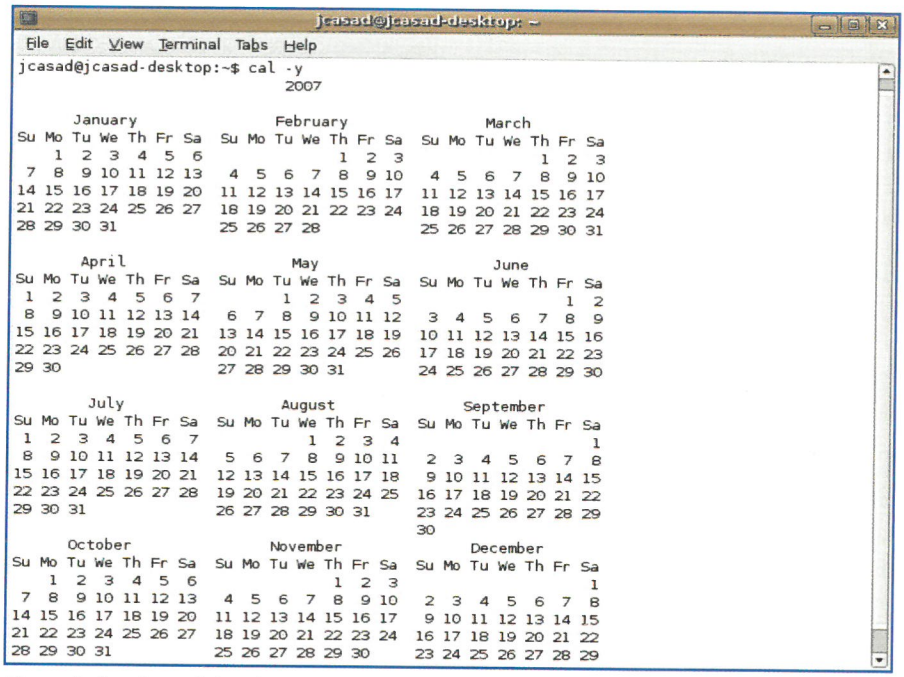


Figura 2: Gracias a Cal podemos disponer de un calendario anual en nuestra consola.

```
$ dateo
Thu Nov  1 15:02:59 CET 2007
```

date también coge como referencia la variable *LANG* para configurar el lenguaje, como cal, y también se puede conseguir la fecha en castellano estableciendo *LC_TIME*, al igual que Cal:

```
$ LC_TIME="es_ES"
dateo.txt.mac.txt
jue nov  1 15:02:38 CET 2007
```

La fecha es incluso más flexible si configuras la variable *TZ* (o zona horaria) con el comando. Comprobando nuestro directorio */usr/share/zoneinfo/* descubriremos los valores de la zona horaria que soporta nuestro ordenador con *TZ*.

Para conocer por ejemplo la hora en Nueva York simplemente ejecutamos el siguiente comando:

```
$ TZ=America/New_York date
Thu Nov  1 10:05:18 EDT 2007
```

¡Oh, sí! Lo queremos en castellano...

```
$ LC_TIME="es_ES"➥
TZ=America/New_York date
jue nov  1 10:06:26 EDT 2007
```

Si nos mudamos a Australia, pongamos el caso, y necesitamos telefonear a algunos amigos en Nueva York de manera regular, podemos configurar un alias para el último comando, evitando así tener que despertar a la gente en mitad de la noche.

Para configurar un alias añadimos la línea siguiente a nuestro fichero bash de configuración, *~/.bashrc*:

```
alias NY='TZ=America/➥
New_York date'
```

y reanalizamos las configuraciones después de guardarlas mediante el comando *source ~/.bashrc*. A continuación, sólo escribiendo *NY* obtendremos la hora de Nueva York en la línea de comandos.

Salida Formateada

El programa *date* posee un importante número de parámetros que influencian el formato de salida. Podemos formatearlo con un signo más (+) seguido por un signo de porcentaje (%), con una letra. Por ejemplo:

```
$ date +%Y_%m
2007_04
```

La tabla 1 lista algunas de las opciones más comunes y puedes utilizar la página man (*man date*) para obtener una lista completa.

Estas opciones de formateo son particularmente prácticas si usamos date para generar automáticamente nom-

bres de ficheros hechos de fecha y/o valores horarios.

El comando que sigue crea un tarball comprimido con Bzip2 con un nombre compuesto de la cadena de texto *backup_*; la fecha, que es el día, el mes y el año separados por guiones bajos; y la extensión del fichero *.tar.bz2* (por ejemplo, *backup_01_11_2007.tar.bz2*).

```
tar -cvjf backup_$➥
(date +%d_%m_%Y).tar.bz2 *
```

Configuración de la Hora del Sistema

El usuario root (sólo el usuario root) puede usar date para configurar la hora y la fecha a una máquina.

Para ello, introduce una cadena de tiempo absoluto añadiendo al comando date los datos como un número de 12 dígitos, dos dígitos para el mes, dos para el día, dos para las horas, dos para los minutos.

Los cuatro dígitos del año van al final:

```
% date 060916101973
Sat Jun 9 16:10:00 CET 1973
```

Las tres primeras partes son obligatorias; si se olvida el año, *date* cargará por defecto el año actual.

Todo es Relativo

Si se desea configurar la fecha con precisión de segundos, Date también incluye valores relativos e incluso tiene un par de cadenas predefinidas para ayudarnos:

- *yesterday*
- *tomorrow*
- *today*
- *now*
- *sec(s)/second(s)*
- *min(s)/minute(s)*
- *hour(s)*
- *day(s)*
- *week(s)*
- *fornight*
- *month(s)*
- *year(s)*

Además, *date* incluye conceptos tales como *ago*, así que podemos decir *1 day ago* en vez de *yesterday*.

Si usamos una de estas cadenas para configurar la hora, deberemos especificar el parámetro *-s* como sigue:

```
# date -s '+3 mins' ➥
Wed Apr 25 18:03:44 CEST 2007
```

Para mostrar una hora relativa necesitaremos hacerlo con el parámetro *-d*:

```
# date -d '+5 days -2 hours'
Wed Apr 30 16:03:44 CEST 2007
```

La página de información de *man* de *date* nos cuenta más acerca de cadenas y de cómo usarlas.

Documentación

Para leer la documentación en la línea de comandos necesitamos usar *info coreutils date* o, en el administrador de ficheros Konqueror de KDE, introducir la URL *info:/coreutils/date input formats*.

Configurar el Reloj del Hardware

Además del reloj de software, nuestro ordenador posee otro cronometrador, y uno que incluso puede mantener la cuenta de los días cuando el ordenador se encuentra apagado o incluso desconectado de la corriente eléctrica.

Para asegurarnos un cronometraje sin interrupciones, las tarjetas principales del ordenador poseen un reloj con una batería de memoria intermedia conocido como reloj CMOS, RTC (Real Time Clock, o Reloj de tiempo real), reloj BIOS e incluso reloj hardware.

Programa hwclock

El programa *hwclock* nos permite leer y configurar el reloj hardware; todos los comandos requieren privilegios root.

Usado en combinación con la opción *-r* puede presentar la hora del hardware local tal y como sigue:

```
# hwclock -r
Wed 25 Apr 2007 ➥
19:24:26 CEST ➥
-0.435565 seconds
```

Adicionalmente, hwclock posee opciones para configurar la hora del sistema para reflejar la hora del reloj hardware (*hwclock -s*) o viceversa (*hwclock -w*).

Una combinación de *--set* y *--date* establece una hora específica. Deberemos introducir una cadena para describir la nueva fecha y hora después del parámetro *--date*.

El formato es exactamente el mismo que la opción *-s* del programa date.

Los comandos:

```
# hwclock - - set ➥
- - date="+2 hours "
```

configuran el reloj hardware con una diferencia de dos horas en el futuro.

¡Automatizado!

Network Time Protocol (NTP) es un estándar para automatizar la sincronización de relojes en nuestros sistemas. La señal horaria se propaga a través de la red desde un servidor NTP a los clientes. Podemos configurar la hora cuando nuestro cliente NTP de la máquina Linux contacta con un servidor en la red.

Esto puede ocurrir durante el arranque, cuando nos conectamos a Internet, o al usar un comando manual en la shell.

Distribuciones

Distintas distribuciones poseen métodos distintos para instalar y configurar NTP. OpenSUSE guarda el software en un paquete llamado *xntp*. Para Debian necesitamos instalar el paquete *ntp*. Los usuarios con OpenSUSE pueden configurarlo usando YaST en la solapa *Network Services*. En el caso de Debian es preciso editar el fichero de configuración */etc/ntp.cpnf*, preferiblemente con un editor de texto. De manera alternativa, como root, podemos usar el comando *ntpdate*.

Cuando se ejecuta la herramienta debemos pasarle el nombre del servidor para contactar, como en:

```
# ntpdate de.pool.ntp.org
25 Apr 23:58:21 ➥
ntpdate[24405]: ➥
adjust time server ➥
85.25.141.60 offset ➥
0.037843 sec
```

Ntpdate podría entrar en conflicto con un demonio NTP ejecutándose en nuestra máquina; en este caso deberemos deshabilitar el demonio primero. Para hacerlo en Debian GNU/Linux nos convertimos en root e introducimos */etc/init.d/ntpstop*. ■

Acceso a dispositivos con mount y fstab

MÓNTATELO

Examinamos las herramientas para montar y desmontar dispositivos de almacenamiento.

POR HEIKE JURZIK

Linux no usa letras para describir los dispositivos al estilo Windows, en su lugar, engancha los dispositivos directamente al árbol de directorios en un proceso conocido con el nombre de montaje, que se lleva a cabo bien en el tiempo de arranque, o bien a través de comandos manuales. Existe un lugar para todo en el árbol del sistema de ficheros de Linux. Los ficheros de dispositivos para dispositivos de todo tipo, tales como dispositivos de red, dispositivos extraíbles o particiones de disco duro, pueden encontrarse típicamente bajo el directorio */dev* (por "device", es decir, "dispositivo").

Nombres

Los nombres de los dispositivos IDE (nombres de discos duros, CD-ROM o dispositivos de DVD) comienzan con *hd* (por "hard disk", es decir, "disco duro"). La letra que sigue a continuación depende del conector y del orden. Cada controlador proporciona dos conectores. El primero de ellos es llamado *principal* y el segundo *esclavo*. Los nombres de los dispositivos son: *hda* (principal) y *hdb* (esclavo) para los dispositivos unidos al primer controlador. Los que cuelgan del segundo controlador son conocidos como *hdc* (principal) y *hdd* (esclavo).

Linux maneja discos SATA, dispositivos de almacenamiento en masa USB y dispositivos ZIP como si fueran dispositivos SCSI. Los nombres de estos dispositivos de fichero comienzan con *sd*. Los CD SCSI y los dispositivos de DVD se conocen como *scd*, y los dispositivos tipo disquetera como *fd*.

Además de las letras, muchos dispositivos tienen también números que reflejan la estructura lógica de los medios de almacenamiento. Por ejemplo, la partición primaria en un disco duro principal IDE unida al primer controlador se cataloga como *hda1*, mientras que la segunda partición sería *hda2*, y así sucesivamente. Los números de las particiones lógicas comienzan en 5. La segunda partición lógica en *hdc* tendría por tanto el fichero de dispositivo mapeado como */dev/hdc6*.

El sistema enumera de la misma manera dispositivos de CD/DVD SCSI (*scd0*, *scd1*, etc.) junto con dispositivos disquetera (*fd0*, *fd1*,etc.), aunque no es muy común disponer de una disquetera en un ordenador moderno.

Numerosas distribuciones usan alias como */dev/cdrom* o */dev/dvd* que indican los nombres de los dispositivos para dispositivos CD-ROM/DVD.

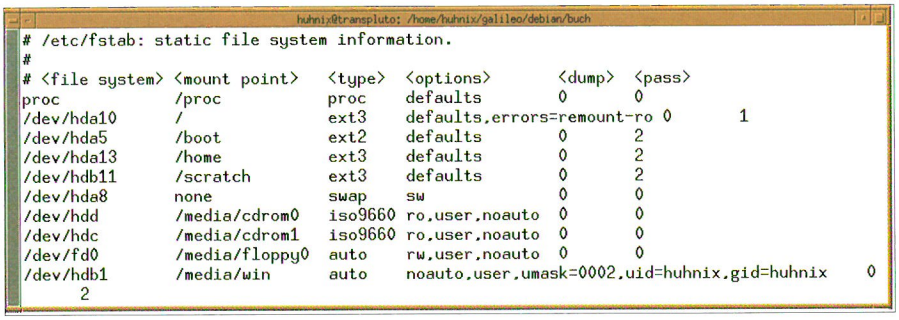
Para mantener el acceso a varios dispositivos y otros sistemas de ficheros hay que crear un enlace entre un dispositivo y un directorio en el árbol del sistema de ficheros de Linux. El dispositivo de fichero está montado de manera que permita que esto ocurra.

Montaje

El montaje ocurre en el momento del arranque o bien manualmente en una etapa posterior. Las particiones del disco duro se montan normalmente durante el tiempo de arranque, mientras que éste es un proceso manual para CDs, DVDs y otros dispositivos extraíbles. El montaje forma parte a menudo del dominio del administrador root del sistema, a menos que este privilegio se haya dado específicamente a los usuarios (ver el párrafo *Tabular, el Fichero /etc/fstab*).

La utilidad que se usa en la línea de comandos para el montaje se llama *mount*. Junto a un determinado número de parámetros opcionales, es preciso especificar el fichero de dispositivo y el punto de montaje. Si se invoca el parámetro *mount* sin especificar ningún parámetro, el comando dirá qué dispositivos se encuentran montados en ese momento:

```
#mount
/dev/hda10 on / type ext3 ➥
(rw,errors=remount-ro)
```



```
huhnix@transpluto: /home/huhnix/galileo/debian/buch
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
/dev/hda10      /               ext3    defaults,errors=remount-ro 0       1
/dev/hda5       /boot           ext2    defaults        0       2
/dev/hda13      /home           ext3    defaults        0       2
/dev/hdb11      /scratch        ext3    defaults        0       2
/dev/hda8       none            swap    sw              0       0
/dev/hdd        /media/cdrom0   iso9660 ro,user,noauto  0       0
/dev/hdc        /media/cdrom1   iso9660 ro,user,noauto  0       0
/dev/fd0        /media/floppy0  auto    rw,user,noauto  0       0
/dev/hdb1       /media/win      auto    noauto,user,umask=0002,uid=huhnix,gid=huhnix    0
       2
```

Figura 1: El fichero fstab proporciona información de las particiones del disco duro y de los dispositivos extraíbles.

```
/dev/hda5 on /boot ➲
type ext2 (rw)
/dev/hda13 on /home ➲
type ext3 (rw)
/dev/hdd on /media/cdrom0 ➲
type iso9960 (ro.noexec,➲
nosuid,nodev,user=huhn)
...
```

Además de las etiquetas de partición y los puntos de montaje, *mount* informa sobre los sistemas de ficheros para los dispositivos, lo cual permite conocer las opciones de montaje que están en ese lugar. La partición */dev/hda10* ha sido formateada con Ext3, por ejemplo, y montada como partición root (en /). El dispositivo de CD IDE contiene un medio con el sistema de ficheros ISO9660 (el sistema de ficheros por defecto para datos de CDs), y ha sido montado en */media/cdrom0*. El listado también nos dice que las particiones del disco duro son de lectura y escritura (*rw* por "read-write"). La opción siguiente:

```
errors=remount-ro
```

asegura que el medio sea remontado de lectura solamente en caso de un error en el sistema, es decir, los datos serán aun de lectura, pero no se dispondrá de acceso de escritura.

Montaje de Dispositivos Extraíbles

Los datos de DVDs, CDs, disquetes y dispositivos USB normalmente se montan manualmente, a menos que se disponga de un automontador que lleve a cabo esta tarea. Linux asigna directorios bajo */mnt* o */media* para dispositivos extraíbles. En la línea de comandos es necesario escribir el nombre del fichero del dispositivo y el punto de montaje. Si se está montando un dispositivo de almacenamiento en masa USB, tiene sentido comprobar el fichero log */var/log/messages* para ver si el fichero ha sido detectado correctamente y para descubrir el nombre de fichero del dispositivo (ver el cuadro titulado *Mensajes Kernel para Sticks USB*).

Para montar el dispositivo detectado aquí, *sdd*, en un directorio existente, */media/usb*, se escribe lo siguiente:

```
mount /dev/sdd /media/usb
```

No es necesario especificar la partición, ya que el stick USB en nuestro ejemplo no está particionado, si bien será preciso especificar */dev/sdd1* para un stick formateado.

Linux auto-detecta el tipo de sistema de ficheros para un medio. Si en su lugar se obtiene un mensaje de error, puede echarse una mano a Linux y especificarle explícitamente el sistema de ficheros, suministrando un valor para el parámetro *-t*, por ejemplo,

```
mount -t vfat /dev/sdd ➲
/media/usb
```

para un viejo sistema de ficheros Windows en un medio con formato FAT. Además de *vfat* (para el sistema de ficheros Windows/DOS), los valores soportados son *ext2* (Extended Filesystem, Versión 2), *ext3* (Extended Filesystem, Versión 3), *reiserfs* (Reiser Filesystem), *iso9660* (ISO9660), *ntfs* (NT Filesystem), etc., por citar unos pocos.

La mayoría de los sistemas definen los nombres de los dispositivos y puntos de montaje para CDs/DVDs y disquetes (ver el cuadro *Tabular, el Fichero /etc/fstab*), y esto significa que un comando tal como

```
mount /media/cdrom
```

Mensajes del Kernel para Sticks USB

```
[...]
01 Aug 17 18:14:15 rachel kernel: Initializing USB Mass Storage driver...
02 Aug 17 18:14:15 rachel kernel: scsi6 : SCSI emulation for USB Mass Storage devices
03 Aug 17 18:14:15 rachel kernel: usbcore: registered new interface driver usb-storage
04 Aug 17 18:14:15 rachel kernel: USB Mass Storage support registered.
05 Aug 17 18:14:20 rachel kernel: scsi 6:0:0:0: Direct-Access     USB DISK 28X    PMAP PQ: 0 ANSI: 0 CCS
06 Aug 17 18:14:20 rachel kernel: sd 6:0:0:0: Attached scsi generic sg2 type 0
07 Aug 17 18:14:22 rachel kernel: sd 6:0:0:0: [sdb] 4030464 512-byte logical blocks: (2.06 GB/1.92 GiB)
08 Aug 17 18:14:22 rachel kernel: sd 6:0:0:0: [sdb] Write Protect is off
09 Aug 17 18:14:22 rachel kernel: sd 6:0:0:0: [sdb] Assuming drive cache: write through
10 Aug 17 18:14:22 rachel kernel: sd 6:0:0:0: [sdb] Assuming drive cache: write through
11 Aug 17 18:14:22 rachel kernel: sdb:
12 Aug 17 18:14:22 rachel kernel: sd 6:0:0:0: [sdb] Assuming drive cache: write through
13 Aug 17 18:14:22 rachel kernel: sd 6:0:0:0: [sdb] Attached SCSI removable disk
```

puede ser todo lo que se necesite para montar un CD.

Opciones de Montaje Críticas

El programa mount dispone de un interesante número de parámetros. La opción *-o ro* hace que el dispositivo sea de "sólo lectura". Para "lectura-escritura" utilizaremos *-o rw*, que es la opción predeterminada. También puede soportar combinaciones: para eliminar accesos de escritura para un medio montado con acceso de lectura-escritura habrá que suministrar dos parámetros cuando se ejecuta el comando , por ejemplo:

```
mount -o remount,ro➥
 /media/usb
```

Esto le dice a mount que *remonte* el medio mientras que al mismo tiempo se deshabilita el acceso de escritura (*ro*).

El comando *mount* posee un parámetro práctico que permite montar imágenes ISO. Para montar una copia 1:1 de un disco antes de copiarlo en un CD/DVD con el propósito de testearlo, se introduce lo siguiente:

```
mount -o loop ficchero.iso ➥
/media/tmp
```

Esto le dice a Linux que use un dispositivo cerrado o de loop para acceder a la imagen como si fuese un dispositivo genuino.

Tabular, el Fichero *"/etc/fstab"*

Como se ha mencionado anteriormente en este artículo, Linux suele montar algunos sistemas de ficheros directamente en el momento del arranque. El fichero */etc/fstab* tiene entradas para los sistemas de ficheros a montar. Además del juego completo de particiones del disco duro, este fichero contiene definiciones para varios dispositivos extraíbles (Figura 1).

La primera columna contiene los ficheros de dispositivo, mientras que la segunda proporciona los puntos de montaje. Las otras entradas especifican el sistema de ficheros para los dispositivos (normalmente el kernel autodetecta esto con *auto*) y varias opciones de *mount*.

Además de los parámetros a los que se ha hecho referencia anteriormente, *ro* y *rw*, podrán verse a menudo entradas como *user* (el dispositivo puede ser montado sin privilegios root), *nouser* (lo opuesto), *auto* (el sistema de ficheros se monta en el momento del arranque), *noauto*, *exec* (los programas en este medio son ejecutables), o *noexec*. Tal y como se muestra en la Figura 1, los dispositivos del CD-ROM, DVD y de disquete en nuestro ejemplo no han sido montados en el momento del arranque (*noauto*), y cualquier usuario puede montar estos dispositivos manualmente (*user*). Como los puntos de montaje apropiados (por ejemplo, */media/cdrom0*, */media/floppy0*) están definidos, cualquier usuario puede escribir

```
mount /media/cdrom0
```

para montar un CD.

Como los sistemas de ficheros Windows no soportan propietarios o grupos de ficheros, es posible (y, de hecho, no es mala idea) especificar un propietario por defecto (*wid* =) y/o grupo (*gid* =) para directorios o particiones de este tipo. Además, la entrada *unmask* en nuestro ejemplo asigna por defecto privilegios de acceso. Para modificar el fichero */etc/fstab* será necesario hacerlo como root y abrirlo en nuestro editor de texto favorito.

¡Fuera!

El comando utilizado para desmontar ficheros es *umount*. Aunque Linux desmonta todos los dispositivos automáticamente cuando se apaga el ordenador, también podrá hacerse manualmente. Así, pueden desmontarse dispositivos extraíbles como CD-ROMs/DVDs, disquetes y dispositivos USB, por ejemplo, usando lo siguiente:

```
umount /media/usb
umount /media/cdrom0
...
```

Es importante asegurarse de que los dispositivos USB y de disquetes han sido desmontados de manera apropiada antes de extraerlos. Sin embargo, los dispositivos de CDs y DVDs se bloquean automáticamente, rechazando ser abiertos cuando el medio está siendo montado.

Existe un mecanismo de seguridad adicional: *umount* no desmonta un sistema de ficheros mientras que un proceso está accediendo a los ficheros:

```
umount: /media/cdrom0: ➥
device is busy
```

Un programa podría estar usando los datos en el dispositivo del CD, o los datos podrían ser parte de un directorio de trabajo usado por la Shell o por un fichero de administración, */media/cdrom0* (o uno de sus subdirectorios). Hay un truco que puede ser usado en la línea de comandos y que permite determinar los procesos que están bloqueando el dispositivo: ejecutando el programa *lsof*, el cual presenta los ficheros y directorios abiertos.

Trabajando como root pueden descubrirse los procesos bloqueados ejecutando *lsof* con el nombre del dispositivo, tal y como se muestra a continuación:

```
# lsof /dev/hdc
COMMAND PID USER  FD  TYPE ➥
DEVICE SIZE NODE NAME
kdeinit 4466 huhn  12r BLK ➥
22,2846 /dev/hdc
```

La herramienta *ps* puede dar información sobre el número del proceso (PID en el listado). La mejor opción es conseguir una larga lista conteniendo todos los procesos en modo de presentación completa, redirigir la salida a la herramienta *grep* y buscar la salida para los procesos ID:

```
@ps auxwww | 4466
huhn 4466  0.0  2.4 27972 ➥
12572 ?  S  11:33  0:00 ➥
kdeinit: kio_audiocd audiocd➥
/tmp/ksocket-huhn/klauncherz➥
9ZRha.slave-socket➥
/tmp/ksocket-huhn/➥
konqueror-socket➥
VPZ1va.slave-socket
```

En este caso, parece que Konqueror es el culpable. Cerramos la ventana del administrador de ficheros que muestre el contenido del CD, o bien cerramos Konqueror y a continuación se debería poder desmontar el CD introduciendo:

```
umount /media/cdrom0
```

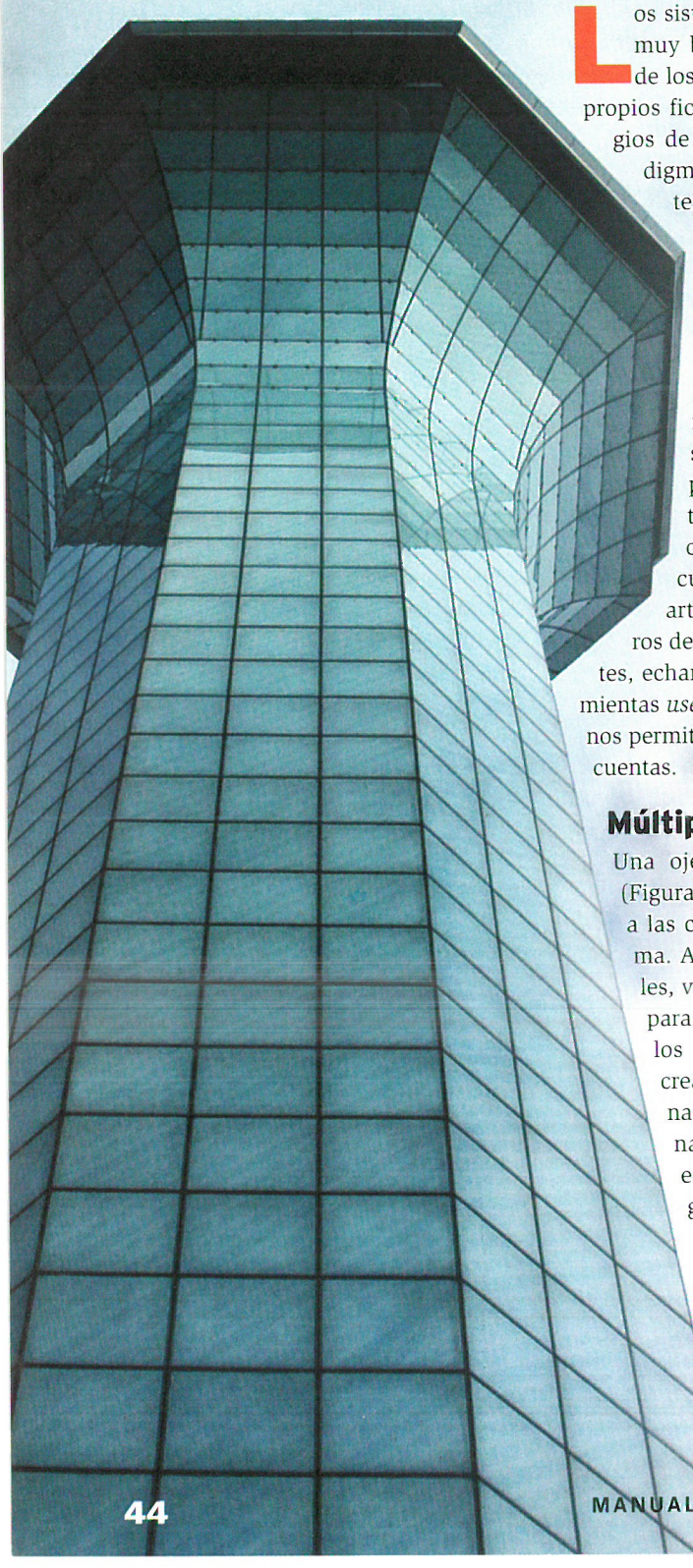
Si esto no ayuda, se puede ser más expeditivo y usar el comando *kill*. ■

Administración de cuentas de usuarios en modo comando

¡ACCESO PERMITIDO!

Linux es un sistema multiusuario: la mayoría de los usuarios pueden trabajar en la misma máquina a la vez en cuentas distintas. Vemos cómo crear y administrar cuentas de usuarios Linux desde la línea de comandos.

POR HEIKE JURZIK



Los sistemas multiusuarios separan muy bien los espacios de trabajo de los usuarios. Pueden salvar sus propios ficheros asignando los privilegios de acceso adecuados. El paradigma típico de Unix/Linux de tener cuentas separadas para tareas administrativas (*root*) ayuda a prevenir la posibilidad de errores de usuarios que podrían destruir el sistema.

El administrador del sistema, *root*, es el mandamás del sistema. Un usuario necesita privilegios de root para ejecutar comandos de configuración y administración de cuentas que veremos en este artículo. Tras mostrar los ficheros de configuración más importantes, echaremos un vistazo a las herramientas *useradd*, *usermod* y *userdel*, que nos permiten crear, administrar y borrar cuentas.

Múltiples Usuarios

Una ojeada al fichero */etc/passwd* (Figura 1) muestra todo lo referente a las cuentas de usuarios del sistema. Además de las cuentas normales, verá que también hay entradas para usuarios como *lp* o *mail*, son los llamados pseudo-usuarios creados por el sistema para asignarles privilegios sobre determinados ficheros y directorios específicos del sistema. Por lo general, las entradas en */etc/passwd* son de la siguiente forma:

```
petronella:x:1002:↵
1002:Petronella↵
  Huhn,,,, ↵
Test-Account: ↵
/home/petronella:/bin/bash
```

Los campos separados por dos puntos contienen las siguientes entradas:

- Nombre de usuario: El nombre que el usuario tecleará para entrar al sistema.
- Contraseña: Aquí es donde va la contraseña encriptada, por defecto en la mayoría de los sistemas aparecerá una *x*, indicando que la contraseña está almacenada en el fichero */etc/shadow*. Si el campo está vacío, el usuario puede acceder al sistema sin proporcionar ninguna contraseña.
- UID: Cada usuario en Linux se identifica de forma unívoca mediante un número o UID ("IDentificación de Usuario"). Al usuario root se le asigna el número 0; desde el 1 al 99 se reservan para cuentas del sistema.
- GID: Cada usuario es miembro de uno o varios grupos, que se identifican mediante el llamado GID ("IDentificación de Grupo"). *Información adicional: Una descripción del usuario, que puede contener varias palabras (normalmente el nombre y apellidos del usuario, un número de teléfono, etc.).
- Directorio home: Para usuarios "normales", generalmente, el directorio home es */home/nombre_usuario;* los pseudos-usuarios tienen su directorio de trabajo (por ejemplo, */var/spool/lpd* para el usuario del demonio de impresión, *lp*).
- Shell: El intérprete de la línea de comandos a lanzar cuando el usuario se registra en el sistema (normalmente */bin/bash*).

Contraseñas en las Sombras

Como se dijo anteriormente, los sistemas actuales tienden a utilizar contraseñas shadow. Estas contraseñas mejoran la

seguridad del sistema eliminando las entradas de las contraseñas encriptadas del */etc/passwd*, que puede leerse globalmente, y colocándolas en */etc/shadow* (el fichero shadow sólo puede ser leído por el administrador). Una característica de las contraseñas shadow es que también almacenan la información sobre el último cambio de contraseña y ofrecen un mecanismo de control a usuarios para cambiar sus contraseñas.

Los campos de las líneas en /etc/shadow también se encuentran separadas mediante dos puntos. En estos campos se almacena la siguiente información:

- *Nombre de usuario*: El nombre de usuario (el mismo que en */etc/passwd*).
- *Contraseña*: La contraseña encriptada. Este campo no puede estar vacío y esto significa que los pseudos-usuarios como *lp* tendrán una entrada con un asterisco "*". Para deshabilitar temporalmente una cuenta, todo lo que necesita hacer es colocar delante de la entrada de la contraseña un signo de exclamación.
- *Último cambio*: La fecha del último cambio de la contraseña, medida en días desde 01.01.1970.
- *Edad Min. / Max.*: Qué antigüedad tiene la contraseña antes de que el usuario la cambie.
- *Aviso*: ¿Cuántos días antes de que caduque la contraseña debe avisarse al usuario?
- *Período de gracia*: Cuánto tiempo se le debe permitir al usuario trabajar después de que su contraseña caduque antes de que se le deshabilite su cuenta
- *Validez*: Fecha en la que caduca la contraseña (medidas en días desde 01.01.1970).
- *Vacío*: El último campo está reservado.

Nótese que sólo las primeras dos entradas para el nombre del usuario y la contraseña son obligatorias; los demás campos son opcionales. Una entrada típica podría ser:

```
petronella:$1$RXbNLkU8➥
$XULLOGFVs6LTxmSRqCS.P/:➥
12872:0:99999:7:::
```

El Entorno Básico

Cuando se crea una cuenta, es normal asignarle unos cuantos valores por defecto a los usuarios. Además al crear un directorio home, podría añadir unos cuantos ficheros de configuración (por ejemplo, para la shell). La mayoría de los sistemas le permiten proporcionar a los usuarios un entorno básico especificado en el directorio **/etc/skel** (para el skeleton).

Mientras Debian Linux sólo almacena unos cuantos ficheros de configuración ocultos (para varias shells) en el directorio skeleton, Suse añade un número de ficheros y carpetas. La mayoría de las herramientas que ayudan al administrador añaden cuentas de usuario automáticamente copiando el contenido de */etc/skel* al directorio home del nuevo usuario y asignando los permisos correspondientes.

Si decide añadir manualmente una cuenta de usuario editando los ficheros */etc/passwd* y */etc/shadow*, asegúrese de crear un nuevo directorio home, copie el contenido del directorio skeleton al nuevo directorio home y modifique los permisos de forma adecuada:

```
mkdir /home/newuser
cp -r /etc/skel /home/newuser
chown -R newuser:users ➥
/home/newuser
```

Para incluir los subdirectorios bajo */etc/skel*, cuando ejecute los comandos *cp* y *chown*, asegúrese de colocar el flag *-r* para la copia recursiva y el flag *-R* para cambiar los permisos recursivamente.

users es el grupo por defecto para nuevos usuarios en la mayoría de las distribuciones; si este no es el caso de su distribución, necesitará reemplazar la entrada *users* por el nombre correcto del grupo.

Investigando useradd

Si todo este trabajo manual es demasiado confuso o le lleva mucho tiempo, useradd es una herramienta rápida de la línea de comandos que puede echarle una mano. Varias distribuciones Linux instalan versiones con parámetros diferentes. Pero no se preocupe: indicaremos las diferencias relevantes en este artículo. Tecleando *useradd* o *useradd -help* le proporcionará una lista de parámetros.

Cada nueva cuenta necesita un Identificador de Usuario (UID), que puede definir mediante el flag *-u UID*. Asegúrese de que el UID es único para cada usuario - si intenta asignar el mismo UID dos veces, *useradd* protestará. Si omite la identificación del usuario, *useradd* determinará el UID más alto en el rango normal de usuario y lo incrementará en 1.

También puede especificar el parámetro *-g* para establecer el GID primario. Si necesita hacer al usuario miembro de cualquier grupo, añada la etiqueta *-G* y el número del grupo apropiado. El fichero */etc/group* le dice qué grupos existen en el sistema y qué GID tienen estos grupos. Vamos a suponer que quiere añadir una nueva cuenta al grupo *users* (p.e., GID 100) y además al grupo *cdrom*, que tiene el GID 24. Las siguientes opciones lo harían: *-g 100 -G 24*.

Por defecto, *useradd* escoge un nombre para el directorio home que refleje el nombre de usuario. Para especificar explícitamente un directorio home diferente hay que especificar la opción *-d directorio*. El parámetro *-m* le indica a useradd que cree el directorio home si no existe y que copie el directorio skeleton desde */etc/skel* al nuevo directorio home. *-c* le permite añadir información para el usuario - nótese que necesita colocar esta información entre comillas si contiene espacios en blanco. La línea de comandos podría parecerse a:

```
useradd paky -u 501 ➥
-g 100 -m -s ➥
/bin/bash -c "Paky Cool"
```

El siguiente paso podría ser teclear *passwd paky* y establecer la contraseña para el nuevo usuario.

Por Defecto

La configuración por defecto hace que *useradd* sea incluso más rápido. Esta configuración puede incluir detalles de grupos, de directorios *home* y *skeleton*; la gran ventaja para el administrador es que se pueden dejar todos estos detalles de la línea de comandos, suponiendo que se quieran mantener los parámetros por defecto.

Para descubrir los parámetros por defecto para *useradd* de Debian Linux y Fedora Core, ejecútese la herramienta con la opción *-D*:

```
debian# useradd -D
GROUP=100
HOME=/home
```

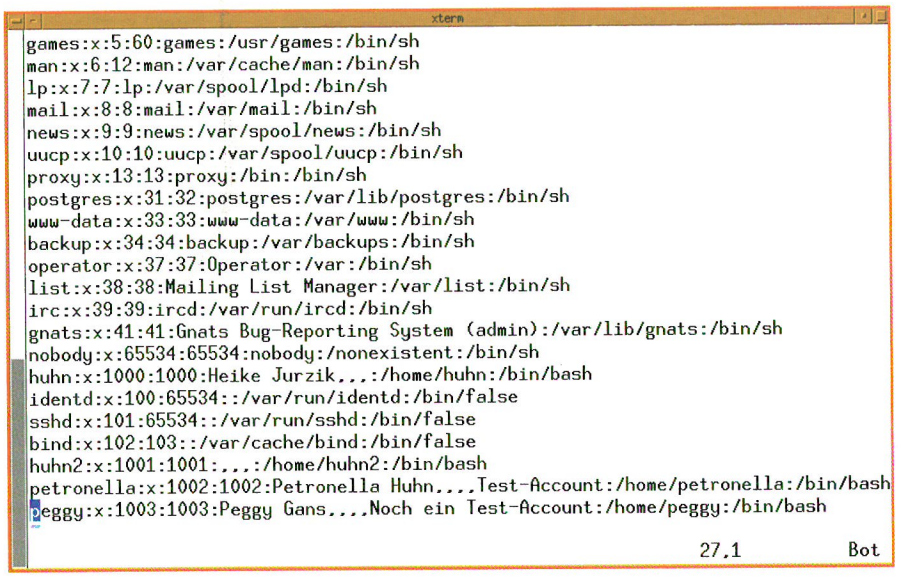


Figura 1: El fichero /etc/passwd tiene detalles de los usuarios del sistema.

```
INACTIVE=-1
EXPIRE=
SHELL=
SKEL=/etc/skel
```

Suse Linux llama a esta opción *--show-defaults*. Suse utiliza los parámetros por defecto para especificar otros grupos a los que se añadirán las nuevas cuentas:

```
suse# useradd --show-defaults
...
GROUPS=video,dialout
```

Si desea que *useradd* utilice los parámetros por defecto, simplemente teclee algo como *useradd < cuenta-nueva >* . Téngase en cuenta que algunas distribuciones Linux esperan que se especifique la opción *-m* si quiere crear un directorio home y copiar el árbol de directorios bajo */etc/skel* a él.

Hay varias maneras de cambiar los parámetros por defecto. Se podría ejecutar un editor y modificar */etc/default/useradd* o podría usar la línea de comandos para definir los valores especificados en los parámetros. Las distintas distribuciones usan diferentes soluciones para este problema. Los usuarios con Debian o Fedora pueden cambiar los parámetros por defecto de *useradd* en la línea de comandos estableciendo el parámetro *-D*. Supóngase que necesita definir una shell diferente. En este caso se teclearía:

```
#useradd -D -s /bin/csh
```

La sintaxis para especificar una localización diferente para los directorios home es similar (*-D -b /home2*), y el comando para especificar un grupo por defecto, diferente (*-D -g 101*).

En Suse Linux, de nuevo se necesita especificar *--save-defaults* en vez de *-D*. Téngase en cuenta que esta opción no funcionará para todos los parámetros por defecto, sino tan sólo para el directorio skeleton (como en *--save-defaults -k /etc/mi_skel*) y la shell (por ejemplo, *--save-defaults -s /bin/csh*. Los parámetros para la modificación del directorio home o GIDs mencionados en las páginas de ayuda no aparecen en las versiones 2.6.90 (Suse 9.2) y 2.6.96 (Suse 9.3), pero siempre se puede modificar */etc/default/useradd*.

Modificando una Cuenta

La herramienta *usermod* permite la modificación de las propiedades de una cuenta de usuario. *usermod* usa funciones y parámetros similares a los de *useradd*. Por ejemplo, la siguiente línea proporcionará al usuario una shell diferente:

```
usermod -s /bin/csh huhn
```

La próxima vez que el usuario huhn se conecte al sistema, él o ella tendrá asignado una shell diferente.

Una de las características más prácticas de *usemod* es su habilidad para deshabilitar cuentas de usuario. Los administradores de Debian que deseen deshabilitar una cuenta de usuario (temporalmente) sin borrarla realmente, pueden teclear el siguiente comando:

```
usermod -L huhn
```

Un vistazo rápido al fichero */etc/shadow* nos dirá que la cuenta realmente ha sido deshabilitada, ya que el hash de la contraseña ahora empieza con un signo de exclamación:

```
huhn:!MjBhdgr.6XkE6:
12575:0:99999:7:::
```

Esta entrada impedirá que el usuario huhn pueda conectarse al sistema. Para rehabilitar la cuenta tecléese:

```
usermod -U huhn
```

¡Adios Usuario!

Es tan fácil borrar las cuentas de usuario como crearlas. La herramienta que necesita para borrar una cuenta de usuario es *userdel*. Hay una condición: que el usuario que se desea borrar no esté conectado al sistema o tenga un proceso ejecutándose en el momento de borrar la cuenta. Suponiendo que este no sea el caso, se puede eliminar la cuenta tecleando *userdel paky*. Si desea eliminar el directorio home al mismo tiempo, especifique el parámetro *-r* para decirle a *userdel* que elimine los datos personales de Paky.

Conclusiones

Aunque la mayoría de las distribuciones tienen herramientas de gestión de usuarios basadas en GUI, la línea de comandos es mucho más rápida a la hora de realizar los cambios si sabe lo que hay que hacer. Una vez que se tenga destreza en el uso de la sintaxis de *useradd*, *usermod* y *userdel*, las tareas de gestión de usuario tan sólo requerirán unos cuantos segundos. ■

LA AUTORA

Heike Jurzik estudió alemán, inglés e informática en la Universidad de Colonia, Alemania. Descubrió Linux en 1996 y ha estado siempre fascinada con el alcance de la línea de comandos de Linux. En su tiempo libre podrá encontrar a Heike escuchando música celta o visitando Irlanda.

chmod, chgrp y chown

ABRIENDO LA PUERTA

Marquis, www.photocase.com

Un sofisticado sistema de usuarios y permisos controla con precisión en Linux quién tiene acceso a qué. Podemos configurarlo todo en la línea de comandos con las herramientas chmod, chgrp y chown. **POR HEIKE JURZIK**

Lo que hace que Linux sea un sistema operativo seguro son los privilegios de acceso granular a ficheros y directorios. Una definición precisa de quién tiene permiso para leer, modificar datos o ejecutar programas específicos proporciona una protección excelente contra los ojos de cualquier fisgón o de una mala configuración intencionada.

El administrador, *root*, no está sujeto a restricciones, lo que incluye asignación de lectura, escritura y la ejecución de permisos a otros usuarios a través del sistema. Si se es el propietario de un fichero o directorio, se puede garantizar el acceso a estos recursos para otras cuentas. Si también se es un miembro de un grupo específico, se puede modificar la propiedad del grupo de ficheros y directorios para las asignaciones de permisos granulares a ficheros.

Consejo

En vez de *ugo*, simplemente se podría poner "todos" (*a*) en *chmod*.

Derechos y Obligaciones

Para cada fichero (es decir, para directorios, ficheros de dispositivo, etc.), Linux define de manera precisa quién puede leer, escribir y ejecutar ese fichero. Además, cada fichero pertenece a un usuario y a un grupo. Los tres permisos se asignan separadamente para esas tres categorías y para usuarios que no pertenecen a ninguna de ellas:

- **Permiso de Lectura:** Los usuarios pueden ver el contenido de un fichero o carpeta en la pantalla, copiarlo o hacer otras cuantas cosas.
- **Permiso de Escritura:** Los usuarios pueden cambiar ficheros y directorios y guardar sus cambios. Esto incluye la habilidad de borrar.
- **Permiso de ejecución:** Para los programas, permisos de ejecución significa que el usuario tiene permiso para ejecutarlo. Ejecutar para un directorio significa que el usuario tiene permiso para cambiar al directorio (adicionalmente, el usuario necesita permiso de lectura para poder ver el contenido de la carpeta).

Descubrir Permisos

Para descubrir los permisos de un fichero podemos ver la carpeta detallada en un

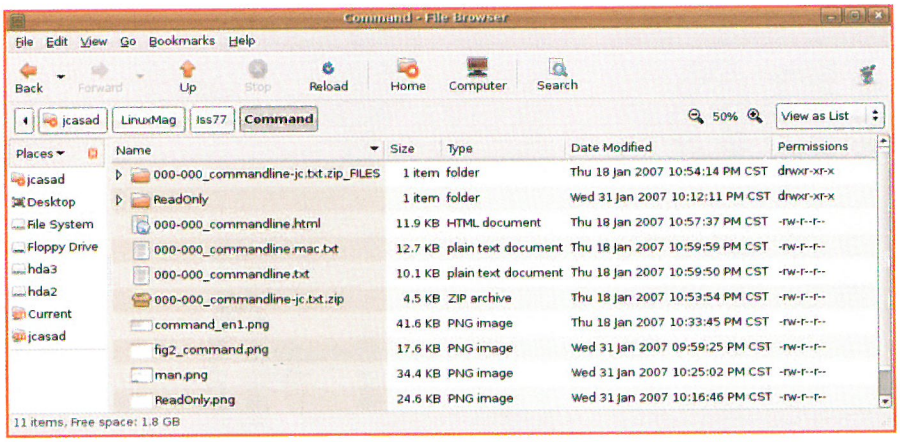


Figura 1: La mayoría de los administradores ofrecen una opción para ver los permisos de los ficheros.

administrador de ficheros gráfico como Konqueror o Nautilus, o simplemente establecer el parámetro *-l* (salida *larga*) para el comando *ls*.

En ambos casos los permisos se indican mediante la letra *r* (de "Read" o lectura), *w* (de "Write" o escritura) y *x* (de "eXecute" o ejecutar). El primer bloque de los tres muestra los permisos para el propietario, el segundo se refiere al grupo y el tercero a todos los otros usuarios. Las carpetas se indican con una *d* (de "directory" o directorio) al comienzo de la lista (ver Figura 1).

Permisos Especiales

Linux posee también dos permisos especiales: el bit *s* (conocido también como el bit *setuid/segid*) y el bit *t* (o "sticky" bit). Ambos sustituyen a la *x* en el bloque de los tres *rwx*.

El *s* se encuentra comúnmente en ficheros ejecutables, mientras que el *t* bit es más común con directorios.

Tal y como sugiere el nombre, el bit setuid/setgid (configuración del usuario ID y configuración del ID del grupo, respectivamente) ejecuta un programa con los permisos de un usuario o grupo sin importar quién ejecute el programa. De este modo, los usuarios sin privilegios pueden acceder a los recursos a los que normalmente no tendrían acceso.

Aunque esto pueda suponer un riesgo potencial de seguridad, sin embargo, el bit *s* tiene sus usos. Muchos programas, incluyendo *su*, *sudo*, *mount* o *passwd* en el ejemplo siguiente cuentan con el bit *s*:

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x root root ➘
27132 Jul 11 20:06 ➘
/usr/bin/passwd*
```

El programa *passwd* modifica las contraseñas, accediendo en el proceso al fichero */etc/shadow* para introducir la nueva contraseña. Por defecto, el fichero se encuentra protegido contra el acceso de escritura mediante usuarios sin privilegios, y reservado para su uso por el administrador para evitar de este modo que cualquiera pueda manipular las contraseñas. El bit *s* ejecuta el programa *passwd* como el usuario root e introduce la nueva contraseña en */etc/shadow* en nombre de root.

El otro permiso especial, el bit *t*, ocurre normalmente en directorios compartidos (lee, escribe y ejecuta permisos para todos), en lugar del parámetro de ejecución, para asegurar que los usuarios sólo tienen permitido modificar, y por tanto borrar, sus propios datos.

El bit sticky está configurado típicamente para */tmp*, tal y como vemos aquí:

```
$ ls -ld /tmp
drwxrwxrwt 16 root ➘
root 4096 Jan
 28 19:51 /tmp/
```

La carpeta */tmp* almacena temporalmente ficheros para múltiples usuarios.

Si todo el mundo tuviera el derecho de lectura, escritura y ejecución de dichos ficheros, en teoría, todo el mundo podría eliminar el sistema y borrar datos arbitrariamente.

Sin embargo, el bit *t* evita que ello ocurra, asegurándose de que los usuarios sólo puedan borrar su propios ficheros (o aquellos sobre los que poseen permiso de escritura). La excepción a esta regla es que el propietario del ficheros con el bit sticky también tiene permiso para borrar dentro de esa carpeta.

Modificación de Permisos

El programa *chmod* permite modificar los permisos de ficheros y directorios asumiendo que eres el propietario o el administrador del sistema, y comprendiendo dos tipos diferentes de comandos.

En un modo, pueden usarse letras para definir permisos. En este caso, *u* para "user" (propietario), *g* para "group" y *o* para "others" (todos los demás usuarios); *r* para "read", *w* para "write", *x* para "execute", *s* para el bit setuid/setgid y *t* para el bit sticky.

Una combinación de estas letras con los signos más, menos e igual le dice a *chmod* que añada, quite o asigne, respectivamente, con precisión estos permisos. Por ejemplo, para dar permisos de lectura y escritura a un grupo para un fichero debemos escribir *chmod g+rw fichero*.

La eliminación de permisos sigue el siguiente método: el comando *chmod o-rw file* elimina todos los permisos para todos los usuarios que no son propietarios ni miembros del grupo propietario.

También es posible combinar ambos comandos como sigue:

```
chmod g+rw,o-rwx fichero
```

Como se ha mencionado previamente, un signo igual permite asignar con precisión todos los permisos especificados en la línea de comandos. Por ejemplo, el comando:

```
chmod ugo=rwx directorio
```

da al propietario, miembros del grupo y a todos los demás usuarios permisos de lectura, escritura y ejecución para el directorio especificado en cuestión.

El programa *chmod* también comprende cifras. Cuando se ejecuta la herramienta, es posible pasar números octales de cuatro dígitos o de tres, en lugar de letras.

Podemos calcular el número como sigue: 4 para permisos de lectura, 3 para permisos de escritura y 1 para permisos de ejecución. El primer número se refiere al propietario, el

Listado 1: Ejemplo

```
01 $ ls -l script.sh
02 -rw-r--r-- 1 huhn huhn 3191789
   Oct 6 05:01 script.sh
03 $ chmod 4755 script.sh
04 $ ls -l script.sh
05 -rwsr-xr-x 1 huhn huhn 3191789
   Oct 6 05:01 script.sh
```

segundo al grupo y el tercero a todos los demás restantes.

Sobre esta base, puedes ver que, por ejemplo, 644 significa *u=rw,go=r* (resultando *rw-r--r--*), o 777 sería igual *a=rwx* (resultando *rwxrwxrwx*). La tabla "Permisos" da más detalles.

Para establecer el bit *s* o el *t* hay que añadir un cuarto número al comienzo del bloque de tres.

El número 4 representa el bit *s* para el propietario (setuid), 2 establece el bit *s* para el grupo (setgid), y 1 establece el bit *t*. El Listado 1 muestra un ejemplo.

Cambio de los Miembros del Grupo

Como usuario "normal" podemos asignar nuestros ficheros propios a grupos específicos; sin embargo, esto ha de implicar que eres un miembro del grupo en cuestión. El comando siguiente nos dice los miembros del grupo:

```
$ groups
huhn dialout cdrom ➥
floppy audio ➥
video
```

Para asignar un fichero al grupo *audio* hay que escribir:

```
chgrp audio fichero
```

Cambio de Propietarios y Grupos

En un sistema Linux, el administrador del mismo puede asignar nuevos grupos a ficheros y directorios. Imaginemos que deseamos configurar una nueva cuenta llamada *mike* y que hemos establecido un nuevo directorio de inicio para Mike y copiados los ficheros de configuración críticos desde */etc/skel*.

El último paso sería dar a Mike los permisos que necesita para establecerse y usar su directorio de inicio y los subdirectorios bajo él.

El comando siguiente entrega el directorio de inicio y todos los ficheros en él (incluyendo los ficheros de configuración oculta) al usuario *mike*:

```
chown -R mike /home/mike
```

La opción *-R* usada aquí le dice a *chown* que actúa de manera recursiva (esto se explicará más tarde). También es útil para poder definir un nuevo grupo propietario para los datos al mismo tiempo:

```
chown -R mike:mike /home/mike
```

En otras palabras, añadimos el nombre del grupo (algunas distribuciones tienen un grupo por defecto llamado *users*, mientras que otras usan el nombre de la cuenta como el grupo por defecto), con una coma para separarlo del nombre de la cuenta.

Recursión

Las tres herramientas, *chmod*, *chgrp* y *chown*, soportan un parámetro *-R* para acciones recursivas. Por ejemplo, si queremos permitir que los miembros del grupo *video* tengan acceso al directorio, a todos los subdirectorios y a los ficheros que contiene, escribiremos:

```
chgrp -R video directorio
```

La opción *-R* también puede evitarnos tener que escribir demasiado cuando la usamos en combinación con el comando *chmod*.

Para eliminar los permisos de lectura, escritura y ejecución de esta carpeta para todos los usuarios que *no* son propietarios o miembros del grupo *video*, escribimos:

```
chmod -R o-rwx directorio
```

Tabla 1: Permisos

Número	Letras Octal
0	---
1	--x
2	-w-
3 (= 2+1)	-wx
4	r--
5 (= 4+1)	r-x
6 (= 4+2)	rw-
7 (= 4+2+1)	rwx

GLOSARIO

Números Octales: El sistema octal usa la base 8; es decir, incluye ocho números entre el 0 y el 7. El siguiente número después del 7 es el 10, 20 sigue a 17, etc. Cada número en un número octal está representado por tres bits; en el caso de los permisos, los tres bits especifican lo que le está permitido hacer a cada clase de usuarios [1].

Aviso

Hay que tener cuidado cuando ejecutamos comandos recursivos que eliminan el parámetro de ejecución. Si erróneamente escribimos *a-x* en vez de *o-x*, descubriremos que nos hemos dejado fuera a nosotros mismos: *chmod* elimina los permisos de ejecución del directorio padre y nuestra habilidad para cambiar al directorio y modificar los ficheros.

El uso de *find* puede ayudarnos a evitar este tipo de dilemas:

```
find directorio -type f -exec ➥
chmod a-x "{}" ";"
```

El comando *find* descubre primero los ficheros (*-type f*) y luego ejecuta *chmod* contra ellos, ignorando el directorio.

Desde el Principio

umask especifica los permisos por defecto asignados a los ficheros y directorios creados recientemente. Si escribimos el comando *umask* sin ningún parámetro, nos presentará la configuración actual:

```
$ umask
0022
```

Lo que vemos es un número octal de cuatro dígitos que especifica qué sustraer de los valores por defecto (0666 para ficheros, 0777 para directorios). En otras palabras, a los nuevos ficheros se les asigna 0644 (*rw-r—r—*) cuando se crean y a las nuevas carpetas 0755 (*rwxr-xr-x*). Para cambiar la umask introducimos el fichero y el nuevo valor en la línea de comandos:

```
umask 0077
```

Esta entrada significa que los nuevos ficheros y directorios solamente están disponibles para sus propietarios. Umask es válido para la shell actual, aunque podemos añadir una entrada a nuestro fichero de configuración bash *~/.bashrc* para hacer que el cambio sea permanente. Trabajando como root, también podríamos añadir una entrada global a */etc/profile* para modificar el umask para el sistema. ■

RECURSOS

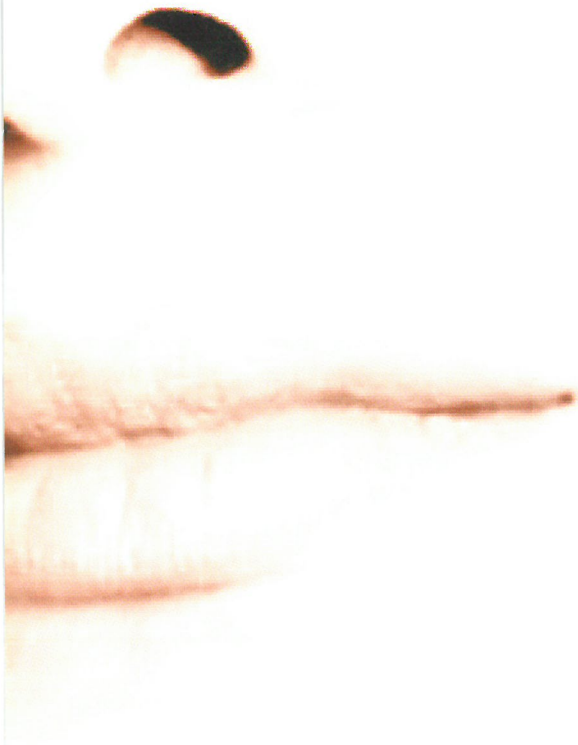
[1] Números octales: *http://en.wikipedia.org/wiki/Octal*

su, sudo

IDENTIDAD

Para más seguridad, aunque tenga privilegios de root para un sistema, tiene sentido utilizarlos sólo temporalmente para prevenir daños accidentales. su y sudo le permiten cambiar la identidad rápidamente desde la línea de comandos.

POR HEIKE JURZIK



Los privilegios de root son necesarios para tareas administrativas, pero no tiene sentido ser el superusuario todo el tiempo. Es preferible convertirse en root para una tarea administrativa y luego volver a ser un usuario "normal". Dos comandos , *su* y *sudo*, le permiten cambiar de identidad.

su

El comando *su* ("substitute user") le permite cambiar su ID en la línea de comandos. Esta instrucción lanza un nuevo intérprete de comandos en segundo plano usando nuevos IDs de usuario (UID) y de grupo (GID). Cuando teclee *su* para convertirse en superusuario, u otro usuario con privilegios, deberá conocer la contraseña para la cuenta de ese usuario.

La sintaxis básica del comando es *su [-] [nombredeusuario]*; pero hay una sutil diferencia, dependiendo de si teclea el signo menos o no . El signo menos (o alternativamente el parámetro *-l* o su forma larga *--login*) se asegura de que realmente entra en el sistema, fijando así las variables de entorno apropiadas y el intérprete de comandos y cambiando al directorio de trabajo de ese usuario (su directorio 'home'). Las variables de entorno no cambiarán si omite el signo menos y esto podría significar que el nuevo usuario no tiene ningún privilegio para el directorio actual (Véase la Figura 1).

Si no suministra un nombre, se supondrá root, la cuenta del superusuario. Esto también es lo que conduce a la falsa idea que *su* es realmente una abreviatura de "superuser". Por omisión, el comando *su* no permite al nuevo usuario lanzar aplicaciones X. Los usuarios externos primero deben tener permiso para utilizar el servidor de X para salida, y esto significa editar el archivo *.Xauthority* en el directorio de trabajo adecuado (véase

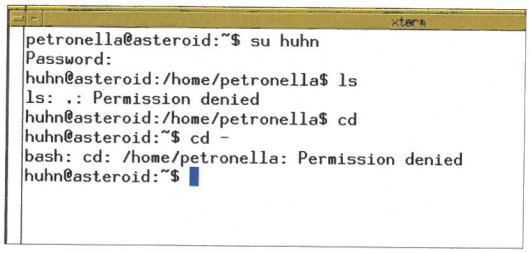


Figura 1: Sin el apropiado ingreso en el sistema, no tiene ningún privilegio.

también man xauth). Para permitir al usuario root lanzar un programa X en un Xterm que pertenezca al usuario *petronella*, necesita extraer una clave desde *.Xauthority*, añadirla al *.Xauthority* del administrador y entonces redefinir la variable DISPLAY (véase el Listado 1).

su también le permite usar otra cuenta para lanzar un solo comando. Para hacer esto, indique la opción *-c (--command)*:

```
huhn@asteroid:~$ su -c"less➥
/var/log/messages"
Password:
```

El uso de la comando *su* se anota en la bitácora (log). En función de la distribución que use, esas entradas de bitácora se localizarán en */var/log/auth.log* (p. e. en Debian) o en */var/log/messages* (p. e. en Suse Linux). Los intentos inválidos son fácilmente localizados, permitiendo al usuario admin ver rápidamente quién ha intentado apropiarse indebidamente de los privilegios de *root*

```
Dec 22 14:50:50 asteroid➥
PAM_unix[2108]: authentication➥
failure;
(uid=500) -> root for su service
Dec 22 14:50:52 asteroid➥
su[2108]: pam_authenticate:➥
Authentication failure
Dec 22 14:50:52 asteroid➥
su[2108]: - pts/8 huhn-root
```

Si es el usuario admin, no necesita introducir una contraseña después de teclear la comando *su*. Usted puede asumir cualquier identidad para probar rápidamente una modificación desde la perspectiva de otro usuario.

sudo

El comando *sudo* le permite evitar dar a conocer la contraseña de root de una máquina, lo cual es comprensible por motivos de seguridad. Realiza lo que el nombre sugiere: "sudo" es la abreviatura de "substitute user, do" (sustituye el usuario, haz) y facilita a usuarios individuales o grupos los privilegios administrativos por un período limitado y limitándose a una tarea especifica. Un usuario puede entonces teclear su propia contraseña para lanzar un comando privilegiado.

El usuario admin necesita crear una lista de usuarios autorizados a ejecutar comandos especificos privilegiados en el archivo */etc/sudoers*. Mientras trabaja como root, edite el archivo con la comando *visudo*. Este programa ofrece las características habituales del editor vi con algunas funciones adicionales. *visudo* "bloquea" el fichero */etc/sudoers* para evitar que sea editado por multiples usuarios a la vez. Además, *visudo* comprueba la sintaxis del fichero al terminar y le informa de cualquier error si lo encuentra:

```
>>> sudoers file: syntax➥
error, line 20 <<<
What now?
```

Dispone de tres alternativas: pulsar *e* para editar el fichero de nuevo, *x* para cancelar los cambios y salir del editor o *Q* para salvar los cambios a pesar del error.

Hay una entrada predeterminada para root *ALL = (ALL) ALL* en */etc/sudoers*. Esto permite hacer todo al usuario root, pero por supuesto root también puede hacerlo sin *sudo*. Si necesita conceder a otro usuario privilegios de root sin restricciones en una máquina, simplemente copie esta línea y sustituya root por el nombre de ese usuario. Después de salvar el archivo, este usuario podrá ejecutar instrucciones de administración mediante *sudo*, por ejemplo:

```
huhn@asteroid:~$ sudo➥
/sbin/shutdown
Password:
```

Si no se permite al usuario usar *sudo*, aparecerá el siguiente mensaje: *"sudo: huhn is not in the sudoers file. This incident will be reported."* (*sudo: huhn no está en el archivo sudoers. Se informará de este suceso*). La medida predeterminada, que puede ser cambiada en */etc/sudoers*, es enviar un correo alertando al administrador con los detalles del usuario que ha intentado lanzar *sudo* (véase la Figura 2). Para estar seguros, los usuarios sin privilegios pueden teclear *sudo -l* para mostrar una lista de los comandos permitidos.

Control Minucioso

La sección *Host alias specification* de */etc/sudoers* le permite especificar las máquinas donde deben aplicarse los comandos específicos de *sudo*. Podemos utilizar el *Host_Alias* para crear un grupo de ordenadores mediante la especificación de sus nombres o definiendo un rango de direcciones IP. Esta característica solamente tiene sentido si aplicamos una configuración centralizada de *sudo* en múltiples ordenadores.

La sección *User_Alias* le permite crear grupos que necesiten los mismos privilegios. Primero definimos el tipo de alias (p. e. *User_Alias*), después un nombre de alias (que puede contener mayúsculas, subrayados y números), un mapeo indicado por el signo " = " y finalmente los nombres de usuario separados por comas. Vamos agregar los usuarios *huhn* y *petronella* a un grupo al que se le permite parar la máquina:

```
# User alias specification
User_Alias SHUTTERSDOWN=➥
petronella,huhn
```

El próximo paso es definir un alias para el comando *shutdown* en la sección *Cmnd alias specification*. Para hacer

GLOSARIO

UID: Cada usuario se identifica mediante un UID ("User IDentification number"), que lo mapea de manera única a la cuenta del usuario. Puede encontrar fácilmente su propio ID tecleando "echo $UID"

GID: Además del UID, los usuarios tienen un GID ("Group IDentification number") que indica su pertenencia a un grupo. Los miembros de un grupo pueden compartir privilegios. La orden "id" indica su UID y su GID actual.

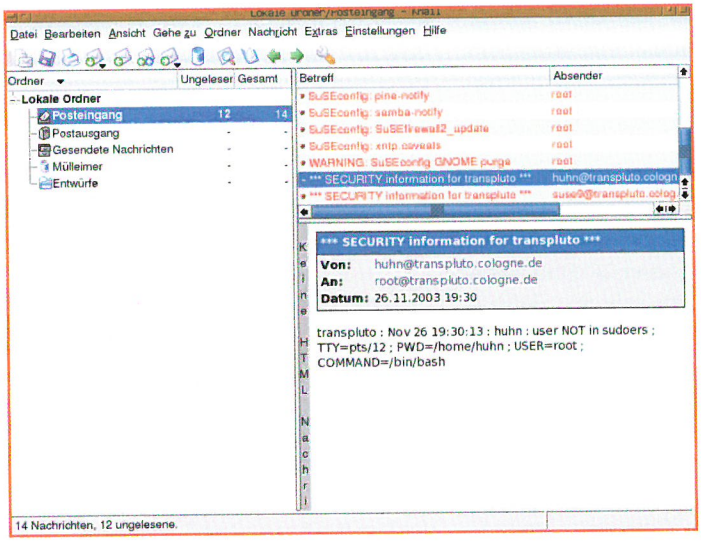


Figura 2: La seguridad es muy valiosa; sudo informa de los accesos no autorizados.

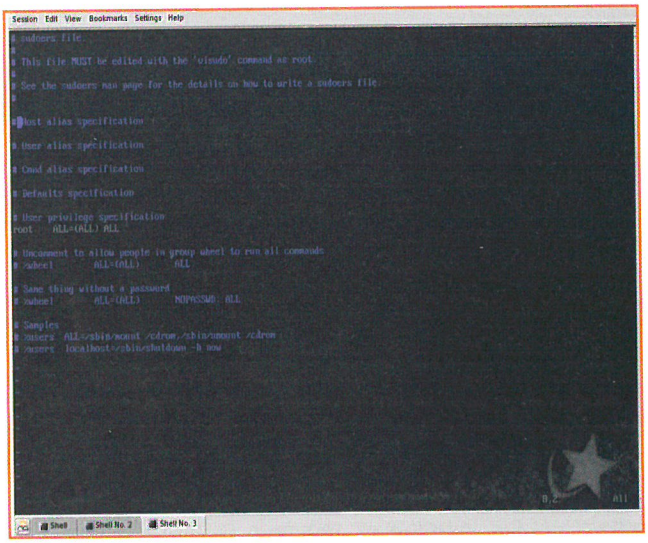


Figura 3: Se recomienda editar el fichero /etc/sudoers solamente con el programa /usr/sbin/visudo.

esto, introduzca la ruta absoluta al programa requerido:

```
# Cmnd alias specification
Cmnd_Alias DOWN =
/sbin/shutdown
```

Para decirle a *sudo* que los *SHUTTERSDOWN* estan autorizados a ejecutar este comando, necesitamos otra entrada debajo de *User privilege specification*:

```
SHUTTERSDOWN ALL = DOWN
```

Los usuarios del grupo *SHUTTERSDOWN* ahora pueden apagar la máquina tecleando *sudo /sbin/shutdown*. Pero hay una forma más fácil de otorgar a un solo usuario permiso para ejecutar un solo comando. Por ejemplo, la entrada:

```
huhn ALL = /usr/sbin/visudo
```

otorga al usuario *huhns* permiso para editar el archivo */etc/sudoers* usando el comando *sudo /usr/sbin/visudo*.

¿Libre o Restringido?

Una simple entrada en */etc/sudoers* le permite restar privilegios a un usuario individual. La sintaxis para hacer esto es como sigue:

```
SHUTTERSDOWN ALL = DOWN
petronella ALL = !DOWN
```

Es importante especificar la excepción inmediatamente después de la regla, ya que el archivo se analiza desde arriba hacia abajo. Esto le permitirá mantener el grupo *SHUTTERSDOWN*, al cual se le puede permitir ejecutar otros comandos, mientras que al mismo tiempo se restringe a *petronella* el permiso para apagar la máquina. Si *petronella* intenta ejecutar el comando, simplemente le mostrará un mensaje como el siguiente: *"Sorry, user petronella is not allowed to execute '/usr/sbin/visudo' as root on asteroid.linux-magazine.com."* (*Lo siento, el usuario petronella no tiene permiso para ejecutar '/usr/sbin/visudo' como root en asteroid.linux-magazine.com*).

Si quiere eliminar el indicador (prompt) de contraseña para una o varios comandos, simplemente active el indicador *NOPASSWD*:

```
SHUTTERSDOWN ALL=NOPASSWD:DOWN
```

En vez de reducir el nivel de la seguridad que sudo proporciona, puede incrementarlo, obligando al usuario que introduzca la contraseña cada vez que ejecute *sudo*. Por omisión, *sudo* ejecuta una especie de sistema de boletos con un intervalo que asegura, por ejemplo, que no pueda abrirse en la máquina una consola del intérprete de comandos con privilegios de root que pueda comprometer todo el sistema sin excepción. La validez predeterminada para el boleto para la mayoría de las distros es de 15 minutos. Pero se puede configurar a 0 minutos añadiendo la siguiente línea a */etc/sudoers*:

```
Defaults timestamp_timeout = 0
```

Listado 1: Display

```
01 petronella@asteroid:~$ xauth
02 extract key $DISPLAY
03 huhn@asteroid:~$ su -
04 Password:
05 asteroid:~# xauth merge
06 home/huhn/key
07 asteroid:~# export
   DISPLAY=:0.0
```

Opcional

sudo también tiene algunos parámetros de línea de comandos. Probablemente el más importante de ellos sea *-s*, que le permite lanzar un intérprete de comandos como root. No es necesario configurar accesos al servidor X, solamente teclear *sudo -s* será suficiente para que el administrador lance programas sobre el servidor gráfico.

El parámetro *-L* lista todas las opciones en el archivo */etc/sudoers*. Si quiere extender su boleto sin ejecutar un comando, solo tiene que introducir sudo *-v*. Si el intervalo se ha agotado, se le indicará que teclee la contraseña. También puede dar de baja un boleto tecleando sudo *-k*. El parámetro *-b* le permite ejecutar un comando en segundo plano; sin embargo, no podrá moverlo de nuevo al primer plano con el comando normal de control de trabajos del intérprete de comandos, *fg*. ■

Configuración de cuotas de disco para usuarios

VECINOS

© Brian Grove - Fotolia.com

En inglés se dice que buenos muros hacen buenos vecinos. Mantener a los usuarios que comparten un mismo sistema en sus propios espacios, bien amurallados, e impedir que allanen los de los demás es una tarea que se puede acometer con el uso de cuotas de disco.

POR PAUL C . BROWN

El escenario es muy común: un servidor de ficheros de repente ya no permite escribir en él o el sistema que lo aloja ya no funciona o no se reinicia. Un poco de investigación revela que los usuarios han llenado tanto el disco duro de fotos y películas, que no tiene ni siquiera espacio para arrancar. Cada nuevo avance en la tecnología de almacenamiento nos da la sensación de que ese nuevo disco duro no lo vamos a llenar en la vida, pero con enormes dispositivos, vienen enormes (y enormes cantidades de) ficheros. En escenarios de uso compartido de recursos, donde, por ejemplo, múltiples usuarios comparten una misma máquina (pensemos en un servidor de ficheros, el servidor de un aula de informática o incluso nuestro cloud interno), es aconsejable establecer pronto cuánto disco duro le corresponde a cada cual.

Un Ejemplo Típico

Imaginemos que hemos vuelto atrás en el tiempo a, digamos... hace un par de años, y tenemos un disco duro de 40 GBs que comparten tres becarios. Para evitar problemas, el administrador toma la decisión salomónica de dividir el disco a partes iguales: 10 GBs para cada usuario y 10 para el sistema.

Lo primero que tendría que hacer nuestro hipotético administrador es comprobar si el kernel de su sistema admite cuotas. Para que un sistema admita cuotas, tendrá que tener insertado en el kernel un driver que nos permita habilitarlas. En muchos casos, este controlador no está cargado. En un sistema Debian, el controlador tiene el siguiente aspecto:

```
quota_vX.ko
```

Donde *X* es un número que suele ser o 1 ó 2.

Si el driver no está cargado (*lsmod | grep quota_v*), podemos cargarlo temporalmente con *modprobe quota_v2*, pero para que se cargue cada vez que se reinicie el sistema, lo suyo es incluirlo en algún fichero en */etc/modprobe.d/*.

Montando la Valla

Una vez que hemos insertado el módulo en el kernel, es hora de hacernos con las herramientas que nos permitirán configurar y administrar cuotas. De nuevo, puede suceder que estas herramientas no están instaladas. Una búsqueda rápida con nuestro administrador de paquetes (*apt-cache search quota* o *yum search quota*) debería devolvernos algún resultado significativo. Tenemos que buscar algún paquete denominado simplemente *quota* e instalarlo.

El siguiente paso consiste en habilitar cuotas en alguna partición editando */etc/fstab*. Las cuotas sólo se pueden imponer en particiones (no en directorios), por tanto, tenemos aquí otra buena razón para separar el directorio */home* en su propia partición a la hora de la instalación: nuestros becarios sólo podrán hacer estropicios – como saturar una partición con vídeos bajados de YouTube – desde su propio directorio, teniendo el resto del disco y particiones vedado. Porque sólo un pésimo e irresponsable administrador de sistemas dejaría a los usuarios campar a

sus anchas fuera de su directorio personal ¿verdad?

Centrándonos, hemos de editar la partición de */home* para que admita cuotas. Si tenemos la línea siguiente en */etc/fstab*...

```
/dev/sda2 /home ext4 ➥
defaults 1 2
```

... o algo por el estilo, lo modificaremos para que tenga el siguiente aspecto:

```
/dev/sda2 /home ext4 ➥
defaults,usrquota,grpquota 1 2
```

No tenemos por qué utilizar ambas opciones, pero *usrquota* habilita cuotas en la partición para usuarios individuales, y *grpquota* las habilita para grupos de usuarios. Tener ambas cosas habilitadas no está de más, sobre todo mirando hacia el futuro: si llega el día en el que, por ejemplo, hemos de establecer cuotas para los miembros del departamento de diseño (que necesitarán mucho espacio en disco para medios gráficos) y del departamento de contabilidad (que no necesitarán tanto), podremos establecer los tamaños de las cuotas en una sola pasada.

Una vez modificado *fstab*, debemos remontar la partición para que los cambios tengan efecto inmediatamente:

```
mount -o remount /home
```

Estableciendo los Límites

Una de las herramientas que instalamos con el paquete *quota*, *quotacheck*, es la que utilizaremos para crear las cuotas propiamente dichas:

```
quotacheck -c /home
```

La opción *-c* es la que le dice a la aplicación que ha de crear las cuotas, y */home* es, como vimos, el directorio para el cual queremos crear las cuotas en este caso. Esta instrucción crea un archivo en */home* que tiene como nombre *aquota.user*.

Curiosamente, *aquota.user* no es un archivo de texto, sino que es binario. Por tanto, para editarlo, no podemos utilizar nuestro *vi*, *emacs* o *nano* de toda la vida. Pero *quota* nos proporciona la herramienta adecuada en la forma del editor *edquota*. La cantidad de opciones admitidas por *edquota* es bastante grande, pudiendo verlas (con una somera explicación de lo que hace cada una) si introducimos *edquota* en la línea de comandos sin ningún otro parámetro adicional.

De momento, lo que a nosotros nos interesa es la opción *-u*. Esta opción nos permite establecer cuotas por usuario, y escribiendo

```
edquota -u mariano
```

por ejemplo, nos permitirá limitar la cuota de disco del usuario *mariano*.

En el Listado 1 podemos ver los diferentes campos que podemos establecer con *edquota*:

- **Filesystem** nos indica la partición en la que nos hallamos trabajando.
- **blocks** indica el número de bloques que ya está ocupando el usuario. Si los bloques son de, por ejemplo, 1024 bytes, eso significaría que mariano ya está ocupando 20 megabytes de disco con sus archivos.
- El primer **soft** indica el límite (en número de bloques) al cual tiene que llegar Mariano para que se produzca un aviso de que se está pasando de su cuota de espacio.
- El primer **hard** indica un límite (en número de bloques) absoluto de espacio al que Mariano jamás podrá exceder.
- Con **inodo** empieza un nuevo bloque de campos que se refieren a eso: los inodos. Hay que entender que podemos limitar el uso del disco por espacio o número de inodos. En este caso vemos que mariano está ocupando 5 inodos con sus archivos y directorios. Como hoy en día es difícil excederse en el número de inodos utilizados, éste es un tema que no tocaremos. Sólo mencionaremos que la sexta y séptima columna (el segundo **soft** y el segundo **hard**) cumplen la misma función que la tercera y cuarta columna, pero para inodos.

Suave como el Terciopelo, Duro como el Diamante

Pero ¿qué queremos decir cuando hablamos de límites "suaves" y "duros"? Pues es bastante sencillo: El límite suave es

Listado 1: Las Cuotas para Mariano

```
Disk quotas for user mariano (uid 1010):
Filesystem          blocks    soft    hard    inodes    soft    hard
/dev/sda2           20        0       0       5         0       0
```

Listado 2: Estableciendo Cuotas para Mariano

```
Disk quotas for user mariano (uid 1010):
Filesystem          blocks    soft    hard    inodes    soft    hard
/dev/sda2           20        9500000 10000000  5       0       0
```

Listado 3: Informe de Cuotas

```
# repquota -a
*** Report for user quotas on device /dev/sda2
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard    grace    used  soft  hard  grace
----------------------------------------------------------------------
root    --  142204  0       0       4       0     0
mariano --  20      9500000 10000000 5       10    20
```

Listado 4: Periodo de Gracia

```
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
Filesystem    Block grace period    File grace period
/dev/sda2     7 days                7 days
```

aquel límite que, cuando Mariano lo cruza, hace que reciba un aviso indicándole que ha excedido su cuota de disco. A partir de ese momento tendrá un periodo (llamado "de gracia", establecido generalmente y de manera predeterminada en siete días, aunque después veremos cómo cambiarlo) en el cual podrá poner en orden su directorio personal, borrando o moviendo ficheros a otro sitio para volver a situar su espacio ocupado por debajo del límite suave. Si en el periodo de gracia no pone en orden su espacio, todo lo que exceda de la cuota establecida se eliminará y no podrá añadir ningún nuevo fichero a su directorio.

El límite duro, por otro lado, es un límite de espacio que no podrá sobrepasar en ningún caso ni en ningún momento.

Si quisiéramos establecer el límite, como vimos antes, en 10 GBs tope para Mariano, eso sería el límite duro, pero podríamos establecer el límite suave en 9,5 GBs, tal y como se ve en el Listado 2 (suponiendo bloques de 1 kilobyte cada uno).

Cuando acabamos de editar las quotas, guardamos el fichero y salimos del editor (Nota: *edquota* utiliza el editor predeterminado del sistema, establecido por la variable de entorno *$EDITOR*. Si el editor predeterminado es *vi*, pulsaremos *:wq* para guardar y salir. Si es otro editor, utilizaremos la combinación de teclas u opciones de menú correspondientes).

Para ver en qué estado queda la cosa, podemos utilizar *repquota*. Sin parámetros, muestra todas sus opciones, con *-a* muestra el estado de todas las cuotas en el sistema, tal y como se ve en el Listado 3.

... Y esto Sólo es el Primer Aviso

Ya sólo nos queda activar las cuotas, y para ello utilizamos la instrucción

```
quotaon -a
```

De nuevo, *-a* sirve para aplicar la instrucción a todas las cuotas. Para ver cómo encender cuotas de algunos usuarios pero no de otros, o activar cuotas por grupo, podemos ejecutar *quotaon* sin parámetros para ver todas sus opciones. La instrucción contraria de *quotaon*, es decir, la instrucción que sirve para desactivar cuotas es *quotaoff*.

Como vimos hace unos párrafos, el usuario, una vez superado el límite suave, tiene un periodo "de gracia" para poner su casa en orden y volver por debajo del límite. Normalmente este límite está establecido en siete días, que comienzan a contar a partir del momento en el que se excede el límite.

Sin embargo, podemos modificar el periodo de gracia con *edquota -t*. El editor que presenta (ver Listado 4) es bastante autoexplicativo, permitiendo establecer el periodo de gracia en segundos (*seconds*), minutos (*minutes*), horas (*hours*) y días (*days*).

Conclusión

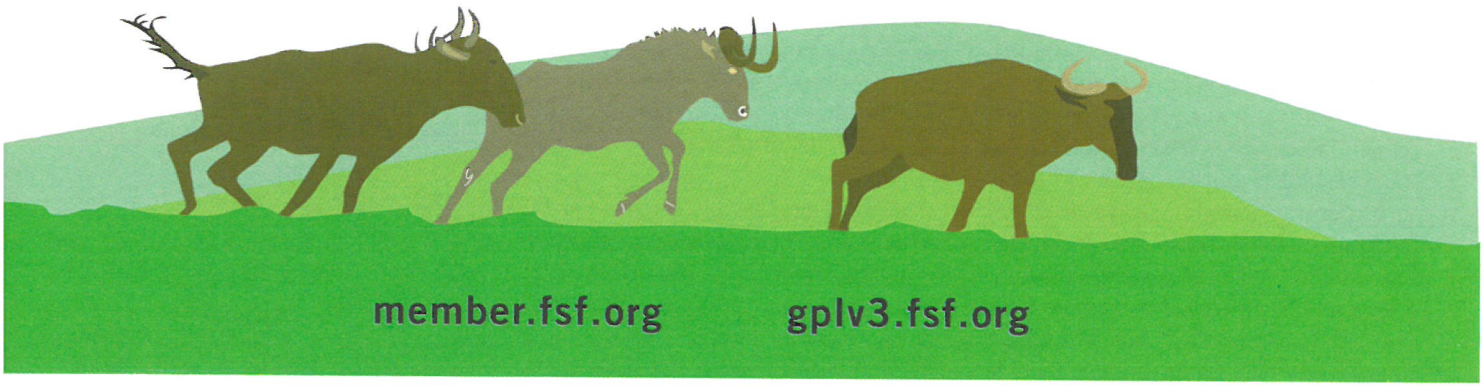
Las cuotas en GNU/Linux, en esta época de discos con tamaños descomunales, es un tema que frecuentemente se ignora... hasta que es demasiado tarde. Pero, con una cada vez mayor cantidad de trabajadores que se conectan desde sus casas a la oficina y usuarios tele-accediendo a sus datos, es vital mantener bien claros los límites entre el espacio de unos y otros si queremos evitar problemas a medio y largo plazo. ■

FREE SOFTWARE FOUNDATION

Freedom:

the freedom to run the software as you wish.

Join the FSF as an associate member.
Your donation will support our 20-year-old commitment to community.



Herramientas de control de procesos y trabajos

ATAREADO

¿Qué tienen en común *free*, *nice* y *killall*? Te descubrimos todo acerca de los procesos de sistemas, cómo monitorizarlos, cómo controlarlos y las herramientas necesarias para ello. Y todo desde la línea de comandos. **POR HEIKE JURZIK**

Siempre que una aplicación o nosotros mismos iniciemos un programa, el sistema operativo comienza uno o múltiples procesos. Dichos procesos pueden adoptar varios estados: Pueden ser procesados, detenidos, reiniciados y – bajo circunstancias normales – detenidos nuevamente. Linux posee algo parecido al control de acceso para controlar procesos: Solamente el usuario que inició el proceso puede detenerlo, restaurarlo o finalizarlo. La única excepción a esta regla es el usuario root, pues puede controlar cualquier proceso en un sistema. En este artículo examinaré las herramientas que monitorizan y controlan procesos.

Listado de Procesos con ps

El comando *ps* nos da una lista de los procesos que están ejecutándose en el momento actual en nuestro sistema. Si no especificamos ningún parámetro en la línea de comandos, la herramienta restringirá la lista a la shell actual. De necesitar más información, podemos especificar alguna de su impresionante colección de opciones. Tal y como nos cuenta su página man, *ps* entiende los parámetros Unix que utilizan un simple guión, las opciones BSD sin guión y las opciones GNU de dos guiones. Elegir una u otra es cuestión de preferencias, aunque en este artículo me centraré en las variantes abreviadas sin guiones.

Si estamos interesados en todos sus procesos, llamamos a *ps* con la opción *x* (Listado 1). La salida tabulada en la shell nos ofrece las siguientes características:

- *PID*: Identificador de procesos. Un único número que referencia un proceso individualmente.
- *TTY*: Terminal/consola en la que se inició el proceso. Un signo *?* indica que el proceso no está ejecutándose en un terminal.
- *STAT*: Estado del proceso. Los estados pueden ser *S* (de sleeping o durmiente), *R* (de running o en ejecución), *D* (de dead, o que el proceso no puede ser reiniciado), o *Z* (zombie, o proceso que ha finalizado sin volver a su estado habitual correctamente).
- *TIME*: Tiempo de ordenador utilizado.
- *COMMAND*: Comando completo con todas sus opciones de la línea de comandos.

El comando *ps* ofrece otras opciones para añadir más información a la salida. Por ejemplo, *u* muestra al propietario del proceso y los ciclos de CPU o porcentaje de memoria, y *a* proporciona una lista de todos los procesos para todos los usuarios.

La opción *l* es también práctica. En su larga salida nos provee de información adicional sobre el PPID (identificador de procesos padre) y del UID (identificación de usuario) del usuario que inició el proceso.

Para presentar lo que pueden ser los parámetros de la línea de comandos bastante largos en la columna *COMMAND*, configuraremos *w* para una salida más ancha, siendo posible utilizar esta opción múltiples veces. Como se muestra en la Figura 1, podemos combinar dichos parámetros cuando sea necesario. El comando *ps* nos permitirá saber si hemos elegido opciones contradictorias para dar formato a la salida.

Una Gran Familia

Los procesos no se encuentran nunca aislados y están siempre en buena compañía. De hecho, se encuentran en una estructura jerárquica, con el proceso número 1, *init*, en lo más alto. *init* es el primer proceso que inicia Linux tras el arranque. Todos los restantes comparten este antepasado común – *init* comienza los programas básicos del sistema operativo. Introduciendo *ps f* nos presenta una jerarquía de procesos en forma de una imagen ASCII. Como alternativa podemos ejecutar el programa *pstree*, el cual

Listado 1: Comando

```
$ ps x
PID TTY      STAT   TIME COMMAND
1358 ?       Sl     0:00 /usr/bin/
gnome-keyring-daemon --daemo-
nize --login
1373 ?       Ssl    0:00 gnome-session
[...]
1634 ?       Sl     0:02 gnome-termi-
nal
1635 ?       S      0:00
gnome-pty-helper
1636 pts/0   Ss+    0:00 bash
1690 pts/1   Ss     0:00 bash
1709 pts/1   R+     0:00 ps x
```

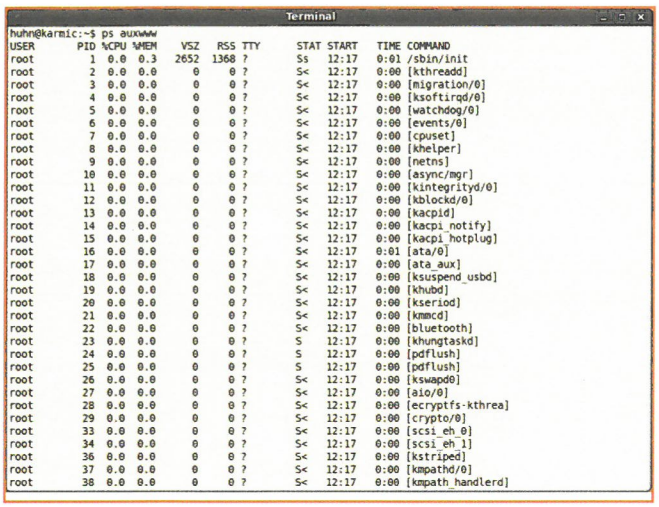


Figura 1: El comando ps nos muestra qué está pasando en nuestra máquina Linux.

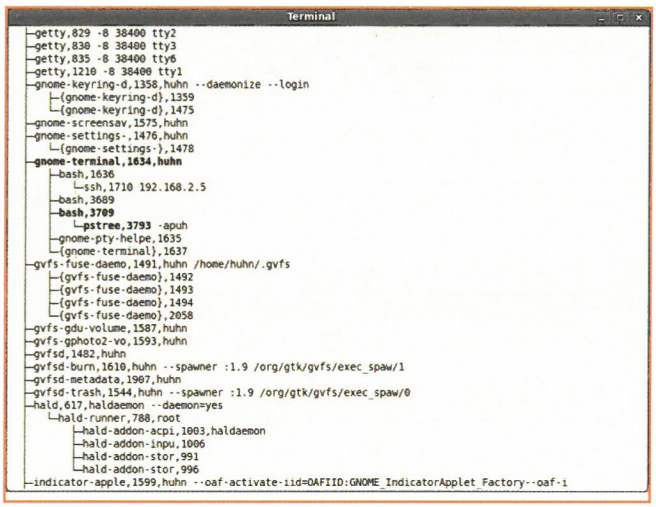


Figura 2: El comando pstree nos muestra las relaciones de procesos en la shell. Combina las opciones para formatear la salida.

nos da una útil perspectiva de las relaciones entre procesos "padre" e "hijo". Esta estructura de árbol nos muestra de un vistazo quién desciende de quién.

La herramienta *pstree* ofrece una salida más detallada si la configuramos con el parámetro *-a*, ya que le dice a *pstree* que nos muestre los parámetros con los que se están ejecutando los programas. Si utilizamos un terminal que soporte diferentes tipos de fuentes y de negrita, como el Terminal de Gnome o la Consola de KDE, probablemente deseemos probar con el parámetro *-h*. Esto le dice a pstree que destaque sus propios procesos y sus ancestros. Si quisiéramos utilizar esta práctica funcionalidad para otros procesos utilizaríamos *-H* con el ID del proceso, con lo que el programa destacará el proceso especificado y su árbol de familia. Configurando la opción *-p* le decimos a pstree que muestre el ID del proceso (PID), mientras que *-u* nos da el usuario. Todos estos parámetros pueden combinarse entre sí – por ejemplo, *pstree -apuh* (Figura 2).

Herramienta de Altura

Si buscamos procesos que consumen mucha CPU, *ps* no es nuestra mejor opción. Como simplemente nos da una captura del estado actual, no descubriremos demasiado sobre la carga del sistema actual. Sin embargo, Linux dispone de la herramienta *top* para ayudarnos en esta tarea. Top es un monitor de procesos que actualiza la presentación para darnos el estado actual. Podemos iniciar el monitor escribiendo *top* directamente en la línea de comandos.

Este programa nos da amplia información sobre nuestro sistema y los procesos que están ejecutándose en él. La línea superior muestra la hora, el tiempo de funcionamiento del ordenador, el número de procesos, y los detalles del estado, junto con la carga de CPU, memoria y swap. Para descubrir más sobre la memoria en uso y la que no está en uso y el espacio swap también podemos usar *free* o *uptime*.

La línea de estado *top* contiene información sobre procesos individuales. Las columnas de la línea de estado presentan varias categorías, como el ID del proceso (PID), el nombre de usuario (*USER*), la prioridad (*PR*), el nivel nice (*NI*, véase la siguiente sección), el uso de memoria en % (*%CPU*) y el nombre del comando (*COMMAND*). Aunque también podemos decirle a *top* lo que queremos ver: pulsamos *F* y las letras pertinentes para especificar el contenido de la línea de estado.

Determinado número de comandos nos permiten controlar *top* interactivamente. Podemos, por ejemplo, pulsar *H* para que se muestre la ayuda en línea. *U* seguido del nombre de usuario nos da los procesos de dicho usuario. [Shift] + [R] nos vuelve a la salida, mostrando los procesos más frugales en vez de los procesos que consumen mucha CPU, e introduciendo *Q* abandona la herramienta y nos lleva de vuelta a la shell.

[Shift] + [Z] nos permite añadir color. La tecla *W* cambia entre algunos esquemas de color predefinidos, aunque también podemos pulsar las letras y números apropiados para definir el nuestro propio (Figura 3).

Un Tipo Amable

Los procesos tienen una prioridad específica, lo cual puede llegar a ser útil si tenemos un programa ejecutándose en el fondo y no queremos arriesgarnos a perder el control sobre la carga del sistema. Para iniciar un programa con una prioridad específica utilizamos el comando *nice*. Los usuarios sin privilegios sólo pueden asignar prioridades bajas a sus propias tareas – la asignación de prioridades más altas es del dominio del administrador.

Los procesos tienen un valor *nice* de *0* por defecto. Si ejecutamos el comando sin parámetros, el programa confirmará:

```
$ nice
0
```

Con el comando *nice* también podemos asignar una prioridad específica, siendo *-20* la más alta y *19* la más baja.

Advertencia

La herramienta *ps* presenta el juego completo de parámetros de la línea de comandos en la columna *COMMAND*. Algunos programas, como el administrador de descargas *wget*, aceptan opcionalmente contraseñas para la autenticación en la shell. La contraseña también aparece en la lista de procesos de *ps*. Teóricamente, cualquier usuario del sistema podría husmear con *ps* estos datos sensibles.

Para establecer el nivel para el monitor del proceso *top*, por ejemplo, escribiríamos:

```
nice -n 19 top
```

Si omitimos la opción *-n* y el nivel *nice*, *nice* configura el valor a *10*. Como se ha mencionado antes, los usuarios normales sólo tienen permitido el uso de valores positivos:

```
$ nice -n -19 top
nice: cannot set niceness: ➥
Permission denied
```

Para descubrir un nivel nice de un programa utilizamos el comando *top* (miramos en la columna *NI* en la barra de estado) o *ps*. En la salida de *ps* en el Listado 2, la llamada a *top* ha sido "niced", como se indica por la mayúscula N en la columna *STAT*.

Para cambiar las prioridades de los programas que ya están ejecutándose usamos el comando *renice*. Los usuarios normales sólo pueden manipular sus propias tareas y sólo el usuario root puede *renice* todos los programas. Para cambiar una prioridad, buscamos el PID del programa, y a continuación usamos *renice* más el parámetro *-n* y el valor:

```
$ renice -n 10 3949
3949: old priority 19, ➥
new priority 10
```

Hablar a Nuestros Procesos

A pesar de que el nombre podría sugerir lo contrario, el programa *kill* necesita no ser fatal. Muy al contrario, se utiliza para enviar señales a los procesos, incluyendo amables peticiones para dejar de funcionar. Como cabría esperar, los usuarios no privilegiados sólo tienen permitido hablar a sus propios procesos, mientras que el usuario root puede enviar señales a cualquiera de ellos.

Si introducimos *kill -l* nos muestra las instrucciones que *kill* pasa a los procesos (Figura 4). Los siguientes son los más relevantes para nuestro quehacer diario:

- *SIGHUP*: le dice a un proceso que se reinicie inmediatamente después de terminar, y con frecuencia se utiliza para decirle a los servidores que parseen ficheros de configuración modificados.
- *SIGTERM*: es una petición para finalizar un proceso limpiamente.
- *SIGKILL*: fuerza a un proceso a terminar, pase lo que pase. Aunque en algunos casos, le lleva más deshacerse del proceso. Después de esperar durante un tiempo, puede que no quede más alternativa que reiniciar la máquina.
- *SIGSTOP*: interrumpe el proceso hasta que introducimos *SIGCONT* para continuar.

Para enviar una señal a un proceso, podemos introducir el nombre de la señal o bien el número seguido del ID del proceso. Por ejemplo, *kill -19 9201*. También podemos especificar múltiples IDs de procesos. Si invocamos a *kill* sin ningún parámetro pero con el PID, enviará la señal *SIGTERM* (*15*) al proceso.

Para encontrar el ID del proceso adecuado ejecutamos *ps* como se ha descrito previamente. El comando de la shell normalmente se combina con otras herramientas, tales como *grep*. Por ejemplo, haríamos lo siguiente para encontrar procesos que contengan *ssh* en sus nombres:

```
ps aux | grep ➥
ssh
```

Además del servidor SSH (*sshd*), la lista incluye todas nuestras conexiones SSH. Para enviar la misma señal a todos estos procesos, listaríamos normalmente los PIDs en la línea de comandos *kill*, lo cual puede ser difícil si la lista es demasiado larga. *killal* nos da una solución: la herramienta entiende todas las señales kill, pero espera nombres de procesos en lugar de IDs. Las siguiente sección explica cómo usar *killal*. También se muestra más información sobre PIDs en el cuadro "El Detective".

Comando Asesino

El programa *killal* soporta las mismas señales que su colega *kill*, pero en vez del ID, espera el nombre del proceso. Si ejecutamos *killal* sin especificar la señal, el programa asumirá que deseamos *-15 (-TERM)*. Por tanto, llamando a *killal gnome-terminal*, finaliza limpiamente todas las instancias de la aplicación del terminal de Gnome.

Como *killal* de hecho realiza un barrido general de los procesos activos, se aconseja precaución. El comando *killal bash*, por ejemplo, finalizará todas las instancias de Bash, incluyendo la shell en la que introdujimos el comando. Sin embargo, podemos especificar la opción *-i* para cambiar a modo interactivo, lo cual nos permite elegir qué procesos matar uno a uno:

```
$ killall -i bash
killall -i bash
Kill bash(1636) ? (y/N) n
Kill bash(3689) ? (y/N) y
Kill bash(3709) ? (y/N) n
```

El comando presenta el PID y nos pregunta por cada proceso que coincide con el nombre que especificamos. En este punto, podemos decidir si permitir que el proceso viva (pulsando la *N*), o si ha llegado la hora de decirle adiós (pulsando la letra *Y*).

Procesos en Primer y Segundo Planos

El algunos casos, un programa que lanzamos en la shell se ejecutará durante un periodo de tiempo amplio. Los programas gráficos que lanzamos en una ventana terminal bloquean la shell, impidiendo la entrada de cualquier comando. En casos como estos, muchos ponen en marcha la ejecución y se van a tomar un café o abren una segunda consola y continúan trabajando. Nosotros somos más listos y moveremos el pro-

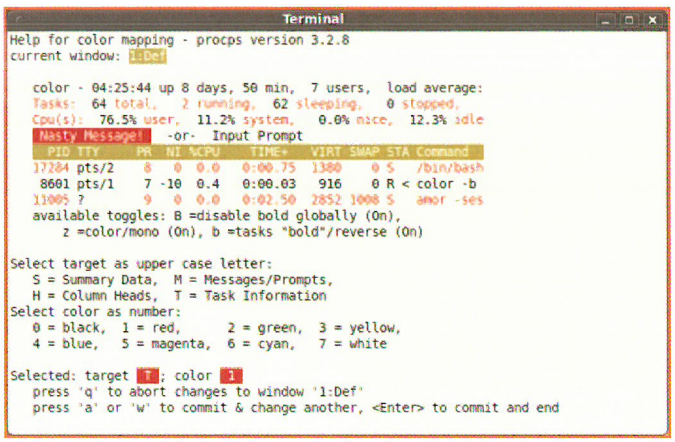


Figura 3: Coloreando la salida del monitor del proceso top.

ceso a un segundo plano, bien cuando lo iniciemos o en un momento posterior.

Para mover un proceso a segundo plano cuando lo arrancamos, simplemente añadimos el carácter ampersand (&) a la línea de comandos. Vemos en el Listado 3, línea 1, cómo iniciar de esta manera a the GIMP. La shell nos informa del ID del proceso (4302) y Bash se libera, pudiendo aceptar más comandos.

Además del ID del proceso, también podemos ver el ID de la tarea entre corchetes (*[]*). El ID de cada tarea se asigna como números consecutivos por la shell. Si lanzamos otro programa en la misma sesión, veremos que Bash asigna el ID 2 a la tarea (Listado 3, línea 4). El comando *jobs* nos dice qué tareas están ejecutándose en la shell actual (Listado 3, línea 6).

Después de que un programa ha completado su tarea, la shell muestra el ID de la tarea junto con su mensaje de estado (*Done*) y el nombre del programa:

```
[1] Done    gimp
```

El ID del trabajo también es útil si necesitamos cambiar un proceso de segundo plano a primer plano o viceversa. Si lanzamos un programa sin que posea un ampersand añadido, podemos pulsar el atajo de teclado [Ctrl] + [Z] para enviarlo a dormir. La shell confirma esta acción como sigue:

```
^Z
[4]+ Stopped ➤
  gimp
```

Si ahora introducimos *bg* (de background), el proceso continuará ejecutándose en segundo plano. El ID de trabajo es útil si tenemos detenidos varios procesos en una shell. El comando *bg %3* le dice al proceso con el ID de tarea 3 que comience a funcionar de nuevo. De manera similar, el programa *fg* (foreground) mueve las tareas hasta el primer plano. De nuevo, este programa puede que necesite detalles de un ID de tarea detrás de un carácter %.

Desconectado

Los comandos que hemos tratado mueven los procesos al segundo plano y, opcionalmente, les permiten continuar ejecutándose. Si cerramos la shell en la que lanzamos el programa, esto también finaliza todos los procesos activos. El programa *nohup* nos da una solución al proteger un proceso de la señal HUP de la shell (véase el cuadro "Hablar a Nuestros Procesos"), permitiéndoles que sigan ejecutándose después de que cerremos la sesión terminal. En otras palabras, esto corta los lazos entre los procesos hijos y sus padres. Para desenganchar los procesos de la shell introducimos *nohup* más el nombre del comando y añadimos un ampersand para enviar el proceso a segundo plano:

```
$ nohup sleep 1000 &
[4] 4497
nohup: ignoring input and ➤
appending output to 'nohup.out'
```

La salida nos dice que el proceso se ejecutará, incluso si escribimos *exit* o si pulsamos [Ctrl] + [D] para abandonar la shell. Un tiempo después, podemos comprobar el fichero *nohup.out* para ver qué hizo el programa mientras estuvimos fuera. ■

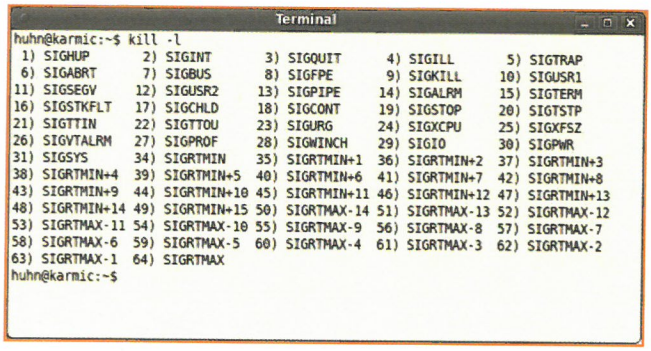


Figura 4: La opción -l presenta una lista de las señales de kill.

Listado 3: Tareas

```
$ gimp &
[1] 4302
$ totem &
[2] 4486
[...]
$ jobs
[1]   Running      gimp &
[2]-    Running      totem &
[3]+  Running      sleep 3600 &
```

Listado 2: Salida ps

```
$ ps auxwww
USER  PID %CPU %MEM  VSZ   RSS TTY      STAT START TIME COMMAND
[...]
huhn  3949  0.5  0.3  2468  1166 pts/2    RN+ 15:32      0:00 top
[...]
```

El Detective

Si estamos buscando IDs de procesos, una combinación de *ps* y *grep* es una buena idea, aunque podemos ahorrarnos tecleos con *pgrep*. Para encontrar los procesos con *ssh* en sus nombres hacemos lo siguiente:

```
$ pgrep ssh
1451
1710
4660
```

Si necesitamos más contexto, añadimos el parámetro *-l* y *pgrep* nos dará los nombres. Para descubrir la línea de comandos completa, incluyendo todos los argumentos, combinamos *-l* y *-f*:

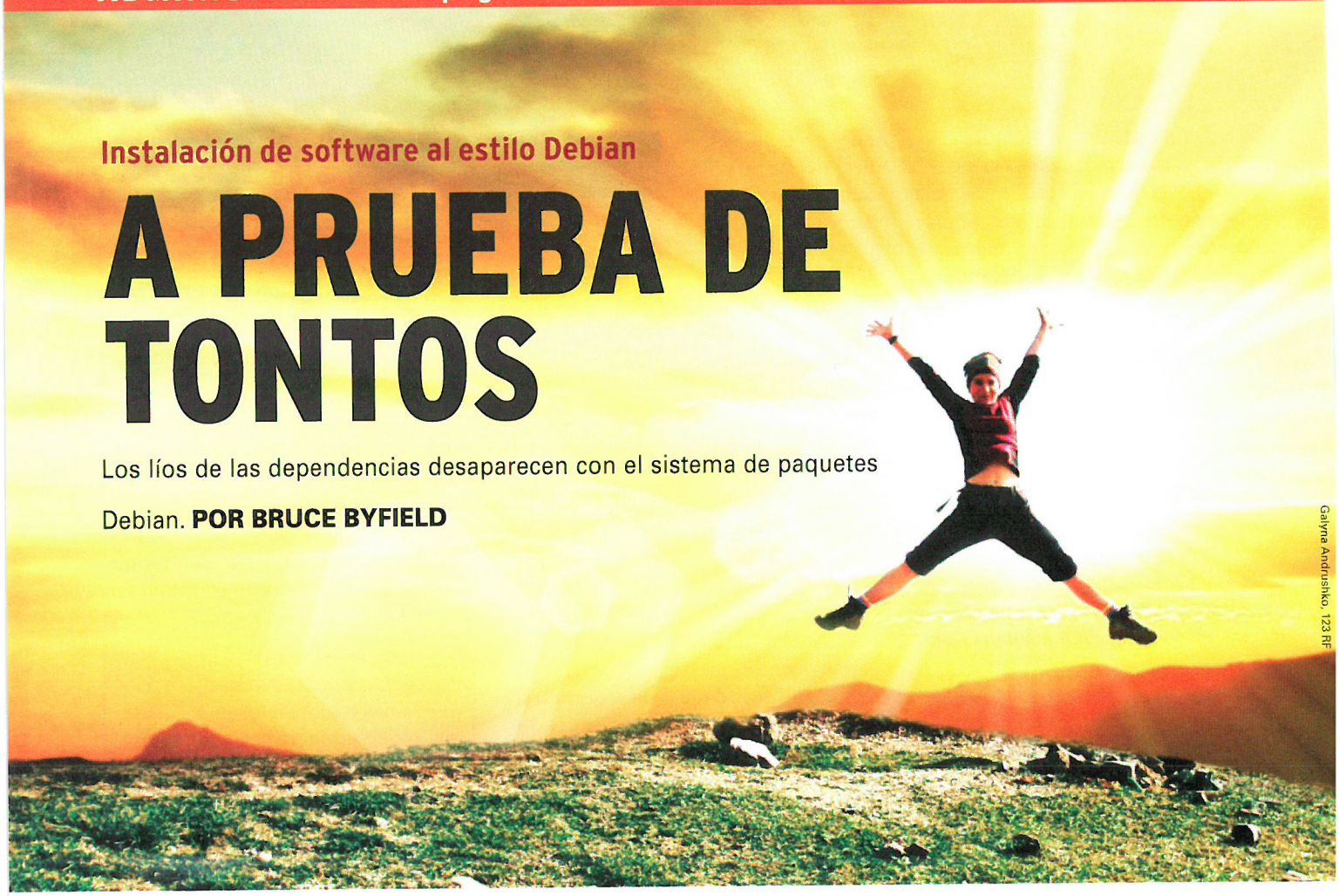
```
$ pgrep -lf ssh
1451 /usr/bin/ssh-agent ...
1710 ssh 192.168.2.5
4660 /usr/sbin/sshd
```

El comando *pkill*, que es una abreviatura para el "comando de asalto" de Linux, entiende las mismas opciones que *pgrep* y se ejecuta contra procesos especificando una señal del mismo modo que *kill*:

```
pkill -19 ssh
```

Otro aspecto práctico es que los administradores pueden querer pasar señales a procesos de otros usuarios con el parámetro *-u*. Para hacerlo, root pasa el nombre de usuario como una opción:

```
# pgrep -lfu petrosilie
7682 sleep 4000000000
7792 bash
[...]
# <B>pkill -19 -u petrosilie<B>
```



Instalación de software al estilo Debian

A PRUEBA DE TONTOS

Los líos de las dependencias desaparecen con el sistema de paquetes Debian. **POR BRUCE BYFIELD**

Galyna Andrushko, 123 RF

Puede que haya oído hablar del "dependency hell", el término utilizado cuando falla la instalación de una aplicación porque necesita una librería de desarrollo que no se encontraba instalada y cuando cada nueva librería que se pretende instalar necesita de decenas de librerías más, reduciendo incluso al linuxero más curtido a un estado de gimoteante frustración.

Afortunadamente, este escenario es en gran parte algo del pasado gracias al sistema de paquetes dpkg y apt-get de Debian. Hace más de una década, los desarrolladores de Debian tuvieron la idea de instalar software dispuesto en paquetes de ficheros que incluían scripts para coger librerías faltantes automáticamente y, de paso, ir configurando sobre la marcha el software. La idea fue copiada finalmente por otros sistemas de paquetes, de modo que ahora nos encontraremos en el infierno de las dependencias sólo cuando intentemos instalar aplicaciones no empaquetadas o que aún están en desarrollo.

El sistema de paquetes Debian se usa en muchas de las distribuciones GNU/Linux más populares. Técnicamente, dpkg es la herramienta que administra el software, mientras que la mayor parte del tiempo los usuarios se relacionan con dpkg a través de apt-get. Ambas se usan para instalar software desde los repositorios online que se describen en */etc/apt/sources.list*.

También podemos añadir */etc/apt/apt.conf*, el fichero que configura apt-get más detalladamente. Sin embargo, la mayoría de la gente prefiere configurar apt-get a través de subcomandos y opciones. Lo básico de estos comandos es fácil de aprender si deseamos añadir o eliminar software o mantener nuestro sistema. Además, encontraremos que en torno a apt-get ha crecido un subsistema completo de utilidades relacionadas para aumentar nuestro control.

Añadir y Eliminar

A diferencia de la mayoría de los comandos, apt-get consta de tres partes separadas por espacios: el comando apt-get básico, un subcomando, y el paquete sobre el que se actúa. Por ejemplo, para instalar el juego solitario Pysol escribimos *apt-get* (el comando básico), seguido por *install* (el subcomando) y *pysol* (el nombre del paquete).

Siempre debe haber un subcomando presente, aunque para determinadas combinaciones – especialmente para mantenimiento – no se necesita una lista de paquetes. Para incluir múltiples paquetes, o se listan separados por un espacio, o bien se usan expresiones regulares, tales como el asterisco, aunque hay que tener cuidado con esto, ya que puede causar problemas y a veces conducir a resultados imprevistos.

En su lugar, si necesitamos instalar múltiples paquetes, lo más acertado será buscar un metapaquete. Un metapaquete es un paquete "de pega" destinado a simplificar la instalación de aplicaciones grandes que están divididas en más de un paquete. Por ejemplo, en Debian, *kde-minimal* instala el número mínimo de paquetes necesarios para ejecutar el escritorio KDE. Para ver si un metapaquete existe, buscamos online en los repositorios de nuestra distribución; si a pesar de todo falla, adivinamos su nombre y vemos si hemos acertado. Con tal de que utilicemos el sentido común, es poco probable que este método tenga consecuencias irreversibles para nuestro sistema.

El comando básico para añadir o actualizar un paquete de software es

apt-get install nombre_de_paquete. Tan pronto como introduzcamos el comando, obtendremos un resumen completo de lo que ocurrirá si llevamos a cabo la instalación, incluyendo las dependencias que se instalarán, los paquetes que se actualizarán y los que se eliminarán, y la cantidad de espacio en disco que se necesitará. A menos que la acción pueda continuar automáticamente sin que afecte a nada más, entonces debemos elegir si continuar con el proceso o no (Figura 1). Es recomendable leer el resumen cuidadosamente antes de continuar para asegurarnos de que lo que introdujimos no incluye sorpresas desagradables.

Si usamos un repositorio online no estándar, puede que no esté verificado automáticamente como una fuente válida. Cuando esto ocurre, hemos de continuar sólo si estamos absolutamente seguros de que podemos confiar en el repositorio.

Cuando apt-get se pone a funcionar, muestra los paquetes que se están descargando y su progreso, además de la velocidad de descarga y del tiempo necesario para finalizar la operación. Los tiempos son sólo estimados y cambiarán cuando cambie la velocidad de conexión a Internet. Una vez completadas las descargas, apt-get instala el software, deteniéndose a veces para realizar preguntas acerca de cómo deseamos que se instale. Una vez que todo esté hecho, si fuera necesario, apt-get presenta un resumen de todos los problemas que ha encontrado. Como toque final, el software instalado es añadido a los menús del escritorio.

El comando básico para instalar software puede modificarse con algunas opciones. Por ejemplo, usaremos *-s* para simular la instalación sin hacer nada, asegurándonos así de descubrir todos los problemas antes de realizar la instalación real. Si esta opción nos informa de algún problema, podemos ejecutar el comando de nuevo, esta vez con la opción *-f*, con la esperanza de que apt-get pueda proporcionarnos de forma inteligente una solución al problema, o con *-m* para ignorar pérdidas de dependencias, con la esperanza de que obtendremos resultados con los que podamos vivir. Si no deseamos responder a preguntas sobre la instalación, podemos utilizar *-y* para responder a todas con "yes" – una opción peligrosa que es preferible evitar si no sabemos qué estamos haciendo. Si queremos re-instalar la misma versión de un paquete, usaremos *--reinstall*.

Sin embargo, quizás la opción más útil para la instalación es *-t repository*, la cual permite especificar el repositorio online desde el que deseamos instalar los paquetes y todas sus dependencias. Esta opción es especialmente útil en Debian, cuyos repositorios principales – estable, de pruebas e inestable – describen el estado del software. Por ejemplo, si deseamos la ultimísima versión de Gnome, incluso sin haber sido testeada, podemos introducir *apt-get unstable install gnome-desktop environment*. De modo parecido, en otras distribuciones basadas en Debian podemos añadir una rama de desarrollo del software a nuestros repositorios o una versión de software desarrollada en privado que sólo usaremos de manera ocasional. Con esta opción podemos descargar un paquete cuando la versión más reciente está en fase beta o sin depurar o no funciona.

```
nanday:~# apt-get install wesnoth
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libboost-regex1.37.0 libboost-iostreams1.37.0
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  ttf-wqy-zenhei wesnoth-all wesnoth-aoi wesnoth-core wesnoth-data wesnoth-dbg
  wesnoth-did wesnoth-editor wesnoth-ei wesnoth-httt wesnoth-l wesnoth-low
  wesnoth-music wesnoth-nr wesnoth-sof wesnoth-sotbe wesnoth-thot wesnoth-tools
  wesnoth-trow wesnoth-tsg wesnoth-ttb wesnoth-utbs
The following packages will be upgraded:
  ttf-wqy-zenhei wesnoth wesnoth-all wesnoth-aoi wesnoth-core wesnoth-data
  wesnoth-dbg wesnoth-did wesnoth-editor wesnoth-ei wesnoth-httt wesnoth-l
  wesnoth-low wesnoth-music wesnoth-nr wesnoth-sof wesnoth-sotbe wesnoth-thot
  wesnoth-tools wesnoth-trow wesnoth-tsg wesnoth-ttb wesnoth-utbs
23 upgraded, 0 newly installed, 0 to remove and 1207 not upgraded.
Need to get 255MB of archives.
After this operation, 11.0MB of additional disk space will be used.
Do you want to continue [Y/n]?
```

Figura 1: Antes de hacer nada, apt-get explica qué hace y nos da la opción de no realizar la operación.

Como alternativa podremos añadir */repository* al final del comando. Sin embargo, esta opción sólo instalará los paquetes especificados desde la fuente especificada, por lo que puede que no funcione bien.

Si somos un experto, podemos descargar un único paquete a nuestro disco duro para la instalación. En tal caso, usaremos dpkg directamente. Por ejemplo, si descargamos una versión de desarrollo del administrador de imágenes digiKam, podremos instalarlo cambiando al directorio que contiene el paquete e introduciendo *dpkg -i digikam*.

Para otras opciones, la estructura del comando es la misma, excepto para el cambio en el subcomando. Incluso las opciones disponibles son las mismas, a pesar de que algunas puedan no tener mucho sentido con cada subcomando. El subcomando *remove* desinstala software, mientras que *purge* elimina todas las trazas de él desde nuestro ordenador (aunque ninguno de los dos elimina dependencias, por lo que puede que necesitemos ejecutar algunos de los subcomandos de mantenimiento listados más abajo). Si deseamos actualizar cada paquete en nuestro ordenador, entonces usaremos el comando *dist-upgrade* en vez de introducir cada paquete individualmente. Para añadir cada aplicación disponible en los repositorios, esté ya instalada o no, podemos utilizar *upgrade*, aunque esta opción no es útil para la mayoría de la gente.

Casi todo el mundo usa el sistema de paquetes Debian para instalar los ficheros binarios precompilados. Sin embargo, si queremos asegurarnos de que todo nuestro software se ejecuta en nuestro sistema tan eficientemente como sea posible, podemos utilizar el subcomando *source* para descargar los paquetes fuente y la opción *-b* para compilarlos en nuestra máquina. Si los fuente requieren dependencias, podemos hacer uso del subcomando *build-dep* para proporcionarlos. Nótese, sin embargo, que la compilación de los paquetes fuente puede llevar un tiempo considerable, particularmente en aplicaciones grandes – puede que incluso sean horas con una aplicación como OpenOffice.org.

Mantenimiento del Software

Para ayudarnos con estas operaciones básicas, dkpg y apt-get incluyen algunas utilidades. Cuando nos topamos con dificultades y estamos buscando información, el comando *dkpg-query* nos la puede dar de forma detallada sobre los paquetes involucrados. Por ejemplo, si escribimos *dkpg-query -p kdepim*, recibimos una descripción del paquete que lista la información de contacto para los desarrolladores que lo mantienen, las

```
nanday:~# dpkg-query -p kdepim
Package: kdepim
Priority: optional
Section: kde
Installed-Size: 68
Maintainer: Debian Qt/KDE Maintainers <debian-qt-kde@lists.debian.org>
Architecture: all
Version: 4:4.3.0-1
Depends: akregator (>= 4:4.3.0-1), kaddressbook (>= 4:4.3.0-1), kalarm (>= 4:4.3.0-1
), kdepim-kresources (>= 4:4.3.0-1), kdepim-wizards (>= 4:4.3.0-1), kmail (>= 4:4.3.
0-1), knode (>= 4:4.3.0-1), knotes (>= 4:4.3.0-1), konsolekalendar (>= 4:4.3.0-1), k
ontact (>= 4:4.3.0-1), korganizer (>= 4:4.3.0-1), ktimetracker (>= 4:4.3.0-1), kdepi
m-strigi-plugins (>= 4:4.3.0-1), kjots (>= 4:4.3.0-1), kpilot (>= 4:4.3.0-1), kleopa
tra (>= 4:4.3.0-1)
Suggests: kdepim-doc
Size: 16052
Description: Personal Information Management apps from the official KDE release
 KDE (the K Desktop Environment) is a powerful Open Source graphical
 desktop environment for Unix workstations. It combines ease of use,
 contemporary functionality, and outstanding graphical design with the
 technological superiority of the Unix operating system.
 .
 This metapackage includes a collection of Personal Information Management
 (PIM) applications provided with the official release of KDE.
Homepage: http://pim.kde.org/
```

Figura 2: La utilidad dkpg-query nos dice todo lo que necesitamos saber sobre cada paquete de software.

dependencias del paquete, tamaño y descripción, además de la página de inicio para el equipo de desarrollo (Figura 2). De modo similar, podemos utilizar la opción *-s* para determinar el estado del fichero o *-L* para ver una lista de todos los ficheros incluidos en el paquete de la aplicación. Toda esta información puede ser de gran valor si nos topamos con problemas, tanto si deseamos resolverlos nosotros mismos como si queremos encontrar a alguien que nos ayude.

El comando apt-get incluye algunas otras utilidades en forma de subcomandos que se utilizan sin referencia a ningún paquete. Al igual que usamos *fsck* para investigar y reparar la estructura de un sistema de ficheros, podemos usar *apt-get check* para asegurarnos de que el sistema de paquetes está en perfecto estado.

Cuanto más instalemos y desinstalemos, más debemos ejecutar apt-get con los subcomandos *clean* y *auto-clean*. El primero de ellos elimina todos los paquetes que hemos descargado e instalado, mientras que *autoclean* elimina todos los que no pueden descargarse más. Ejecutando ambos ocasionalmente, podemos liberar espacio extra en nuestro disco duro sin que afecte al sistema.

Otro subcomando de mantenimiento útil para apt-get es *autoremove*, que elimina paquetes huérfanos (por ejemplo, los que no cumplen ningún propósito porque se añadieron como dependencias para una aplicación que ha sido eliminada con anterioridad). Como estos huérfanos no hacen nada pero ocupan espacio en nuestro disco duro, también es buena idea eliminarlos. A diferencia de *clean* y *autoclean*, el sistema de paquetes Debian lleva la cuenta de los huérfanos y nos recordará que existen cuando ejecutamos apt-get para otro propósito.

Otra opción de mantenimiento es la de añadir o eliminar repositorios de */etc/apt/sources.list*. La edición de este fichero puede realizarse con vim, emacs o en cualquier otro editor de texto. El fichero *sources.list* lista todos los repositorios online que usan apt-get y dkpg. Cada repositorio se lista en su propia línea conforme a un sistema simple. La entrada para cada uno comienza con *deb* si es un repositorio de binarios y *deb-src* si se trata de uno de paquetes fuente. Esta información va seguida de la URL del repositorio, nombre y subsecciones. Las fuentes en principio se encuentran deshabilitadas con una marca almohadilla (#) al comienzo de la línea. Típicamente, estas mismas marcas también son usadas para añadir comentarios que los humanos podemos usar para identificar los fuente.

Después de añadir y eliminar un repositorio de *sources.list*, debemos ejecutar *apt-get update* para cambiar los repositorios que están usando apt-get y dkpg. En caso contrario, el sistema de paquetes Debian continúa usando aquellos previamente identificados. La edición tras la actualización tarda unos cuantos minutos para completar cada vez, aunque posee la ventaja de asegurarse de que conocemos con precisión qué fuentes estamos usando. Por esta razón, algunos usuarios prefieren editar *sources.list* y especificar la opción *-t* para establecer desde qué fuentes se instalan. De este modo, las posibilidades de cometer un error son mínimas.

Utilidades Relacionadas

Dependiendo de nuestra distribución, apt-get y dkpg pueden tener determinado número de utilidades con las cuales están asociadas. Por ejemplo, la mayoría de las distribuciones probablemente tengan apt-cdrom, que puede especificar una fuente para la instalación desde CD o DVD, aunque esta utilidad suele sólo utilizarse en una primera instalación.

Otra utilidad menos común es apt-spy, que puede usarse para determinar el repositorio más rápido para usar. El único problema es que las velocidades de conexión pueden variar dependiendo del número de usuarios, y necesitaríamos ejecutar apt-spy varias veces antes de utilizar sus resultados para editar */etc/apt/sources.list*.

Si somos prudentes, desearemos ver si nuestra distribución incluye apt-listbugs antes de instalar un paquete. Esta utilidad busca cualquier error que se ha notificado para la versión del software que estamos a punto de descargar.

Con diferencia, la utilidad de paquetes más útil es apt-cache, que ofrece un tesoro de información sobre paquetes y sobre nuestro sistema. Por ejemplo, *apt-cache showpkg nombrepaquete* muestra las versiones que hemos instalado, la última versión disponible y los repositorios que estamos usando, y las dependencias inversas de los paquetes (por ejemplo, qué paquetes dependen de ella).

De manera similar, *apt-cache dump* lista todos los paquetes que hemos instalado, y *apt-cache stats* ofrece información tal como el número de paquetes instalados y el número total de dependencias. Una opción especialmente útil es *apt-cache search nombrepaquete*, que localiza el nombre exacto del paquete o los paquetes que querremos instalar.

Configuramos el Ejemplo

La combinación dkpg y apt-get tiene más de doce años de antigüedad. En cierta medida, está comenzando a mostrar su edad, con algunas opciones antiguas que pocos, o ningún usuario moderno utiliza, y con mejoras tendentes a entrar en utilidades relacionadas en vez de en dkpg o apt-get. El resultado es un juego de herramientas que es a veces pesado y al que lleva su tiempo acostumbrarse.

A pesar de ello, la combinación dkpg/apt-get mantiene la potencia si nos tomamos un tiempo en aprenderla. No es sorprendente que, siguiendo su ejemplo, otros sistemas de paquetes, como yum o urpmi, hayan racionalizado también sus procesos de instalación. ■

Yummy, yummy, yummy

PAQUETES SABROSOS

Yum, el administrador de paquetes de RPM, tiene sus propias ventajas sobre otras herramientas.

POR BRUCE BYFIELD

Hace una década, los usuarios de Debian solían mirar con cierto desprecio a los usuarios de Red Hat y de otras distribuciones basadas en RPM. Mientras que los de Debian tenían apt-get y dpkg para salvarles del demonio de las dependencias – situaciones en las que la instalación de software es imposible debido a una librería no instalada – los usuarios de distribuciones basadas en RPM debían quitárselo de encima ellos mismos.

Hoy, los usuarios de Debian y Ubuntu no tienen ninguna razón para ser tan engreídos. Las distros de RPM se han puesto al nivel de las herramientas Debian y producido algunos administradores de paquetes que igualan más o menos la funcionalidad de apt-get.

De dichos administradores de paquetes, el más popular es *Yellowdog Updater, Modified*, más conocido como Yum. Esta reescritura de un temprano administrador de paquetes para Yellow Dog Linux se mantiene actualmente a través de Seth Vidal, un empleado de Red Hat, y es usado por muchas de las principales distribuciones RPM, incluyendo Fedora, Red Hat y CentOS.

Al igual que apt-get en Debian proporciona a los usuarios acceso a la funcionalidad de *dpkg*, del mismo modo Yum actúa como un encapsulador para *rpm*, el comando básico para la administración de paquetes RPM. La principal diferencia es que, mientras que *dpkg* resuelve problemas de dependencias por sí mismo, *rpm* no. Esa funcionalidad reside completamente en Yum.

Desde la perspectiva del usuario, esta diferencia no tiene importancia. A pesar de que podemos utilizar Yum indirectamente desde cualquier aplicación de escritorio como PackageKit y Yumex, Yum está tan bien escrita que podemos aprender a ejecutarla de manera fácil directamente desde la línea de comandos.

```
[root@localhost ~]# yum install sil-gentium-basic-book-fonts.noarch
Loaded plugins: dellsysidplugin2, presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package sil-gentium-basic-book-fonts.noarch 0:1.1-4.fc11 set to be updated
--> Processing Dependency: sil-gentium-basic-fonts-common = 1.1-4.fc11 for packa
ge: sil-gentium-basic-book-fonts-1.1-4.fc11.noarch
--> Running transaction check
---> Package sil-gentium-basic-fonts-common.noarch 0:1.1-4.fc11 set to be update
d
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                        Arch       Version          Repository  Size
================================================================================
Installing:
 sil-gentium-basic-book-fonts   noarch     1.1-4.fc11       fedora     425 k
Installing for dependencies:
 sil-gentium-basic-fonts-common noarch     1.1-4.fc11       fedora      21 k

Transaction Summary
================================================================================
Install       2 Package(s)
Update        0 Package(s)
Remove        0 Package(s)

Total download size: 446 k
Is this ok [y/N]:
```

Figura 1: Yum nos da mensajes constantes sobre lo que está haciendo.

Aprendemos los Básicos

Yum, como apt-get, posee un formato básico consistente: el comando básico (*yum*), el subcomando (lo que queremos hacer) y los paquetes involucrados. La diferencia principal está en la lista de subcomandos que intervienen. Yum es más organizado que apt-get, y algunos de sus subcomandos proporcionan funcionalidades que en apt-get residen en una utilidad relacionada, pero separada, tal como apt-cache.

El subcomando que probablemente usaremos más será *install*. Por ejemplo, si planeamos instalar el tipo de letra Book para la fuente libre Gentium, el comando básico en Fedora sería:

```
yum install ➥
  sil-gentium-basic-book-fonts
```

como sería con apt-get (aunque el nombre exacto del paquete podría diferir).

Sin embargo, Yum es un tanto más prolijo que apt-get a la hora de ofrecer feedback (Figura 1). Comienza listando los plugins que están instalados para Yum, luego calcula qué paquetes necesitan ser actualizados y las dependencias que necesita el paquete requerido. Por ejemplo, si decidimos instalar Gentium Book, Yum apuntará que requiere el paquete *sil-gentium-basic-fonts-common*, el cual es necesario con cualquier peso de Gentium que instalemos.

Finalmente, Yum observa si se encuentran disponibles todas las dependencias y presenta todo lo que necesita ser instalado en una tabla. A ésta le sigue una segunda que lista las transacciones (por ejemplo, pasos) necesarias para completar la instalación y la cantidad total de espacio de disco duro requerido. Solamente entonces nos ofrece la opción de continuar o detener la instalación.

Una vez que pulsamos *y* (sí) para continuar el proceso de instalación, Yum comienza a descargar los paquetes necesarios, mostrando el progreso de cada descarga y del total de procesos. Después de que las descargas se han completado, instala cada paquete y resume lo que ha hecho (un paso útil, en el que la información general podría fácilmente haberse desplazado fuera de la vista). Si tiene éxito, presenta un escueto *Complete!* antes de salir (Figura 2).

Con apt-get usaríamos dpkg para instalar un paquete que habríamos descargado a un sistema Debian; sin embargo, con Yum simplemente usaríamos el subcomando *localinstall* y no tendríamos que cambiar a otro comando básico.

Para instalar una versión más nueva de un paquete también podemos utilizar install, aunque es mejor elección el subcomando *upgrade* porque puede controlar la eliminación de cualquier paquete obsoleto – una funcionalidad que es especialmente útil cuando estamos cambiando de una versión de distribución a otra. Para obtener el mismo resultado podemos usar *yum update --obsoletes*. Si somos cautelosos, puede que prefiramos el subcomando *check-update* para ver que está disponible antes incluso de instalar nada. O, en su lugar, puede que prefiramos especificar paquetes individuales para su actualización.

Para instalar vas versiones paquetes loca- de

les, el subcomando es *localupdate*. Para desinstalar empleamos *remove*.

Todos estos subcomandos básicos se encuentran disponibles para su uso con múltiples paquetes. La manera más fácil de manejar estos últimos es introduciendo un espacio entre ellos al final del comando. Alternativamente, podemos usar expresiones regulares, aunque lo mejor es emplear el subcomando *search* primero para ver qué paquetes estarán afectados.

Algunos repositorios Yum organizan paquetes en grupos. Por ejemplo, en Fedora 11, los grupos de paquetes incluyen Games, KDE Desktop y Publishing. Estos grupos hacen la misma función que los meta-paquetes en sistemas Debian, permitiéndonos instalar múltiples paquetes sin tener que recordarlos o editarlos separadamente. Los grupos poseen una serie de subcomandos especiales que incluyen *groupinstall*, *groupupdate* y *groupremove*, seguido por el nombre del grupo. Por ejemplo, *groupinstall publishing* añadiría todos los ficheros a nuestro sistema en el grupo publishing.

Subcomandos de Información

Además de estos subcomandos básicos, Yum incluye algunos otros que proporcionan información o nos ayudan a mantener nuestro sistema.

El subcomando más básico, *list*, se completa con descripciones auto-explicativas de la información que deseamos. Por ejemplo, el comando *yum list installed* muestra una lista completa de paquetes instalados. De forma similar,

```
root@localhost ~
File  Edit  View  Terminal  Help
Total download size: 2.5 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
fedora/prestodelta                                       |  410 B     00:00
Processing delta metadata
Package(s) data still to download: 2.5 M
(1/2): PySolFC-music-4.40-4.noarch.rpm                   | 1.7 MB     00:02
(2/2): pygame-1.8.1-6.fc11.i586.rpm                      | 882 kB     00:01
--------------------------------------------------------------------------------
Total                                        603 kB/s | 2.5 MB     00:04
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : pygame-1.8.1-6.fc11.i586                                 1/2
  Installing     : PySolFC-music-4.40-4.noarch                              2/2

Installed:
  PySolFC-music.noarch 0:4.40-4

Dependency Installed:
  pygame.i586 0:1.8.1-6.fc11

Complete!
```

Figura 2: Cuando instalamos los paquetes, Yum nos mantiene informados de su progreso. Nótese que está verificando DeltaRPMs, lo que indica que Yum está utilizando el plugin yum-presto para minimizar el tamaño de las descargas.

```
[root@localhost ~]# yum info evolution
Loaded plugins: dellsysidplugin2, presto, refresh-packagekit
Installed Packages
Name        : evolution
Arch        : i586
Version     : 2.26.3
Release     : 1.fc11
Size        : 38 M
Repo        : installed
Summary     : Mail and calendar client for GNOME
URL         : http://projects.gnome.org/evolution/
License     : GPLv2+ and GFDL
Description: Evolution is the GNOME mailer, calendar, contact manager and
           : communications tool.  The components which make up Evolution
           : are tightly integrated with one another and act as a seamless
           : personal information-management tool.
```

Figura 3: El subcomando "info" proporciona información disponible sobre paquetes.

yum list available lista todos los paquetes en todos los repositorios y *yum list updates*, todas las actualizaciones disponibles. Otro descripción, más específica, incluye *extras*, el cual lista paquetes en nuestro sistema que no están listados en repositorios no habilitados, y *recent*, que lista las últimas actualizaciones en los repositorios.

Cuando deseamos información más específica sobre un paquete, el subcomando a usar es *info*, seguido por el nombre del paquete. El comando *info* proporciona información básica sobre el paquete: su arquitectura, número de versión y edición; si está instalado o, si no, en qué repositorio está; su licencia y su página de inicio (Figura 3). También recibiremos un resumen de una única frase y una descripción ligeramente larga. El subcomando *groupinfo* ofrece información similar para grupos de paquetes.

Uno excepcional, aunque ocasionalmente útil, es *provides*. Con el comando *provides* podemos encontrar qué paquete incluye un fichero o funcionalidad específica (Figura 4). Por ejemplo, en Fedora 11, el comando *yum list provides firefox* devuelve exactamente qué versión de paquete está disponible o instalada además de la versión encontrada en los repositorios.

Otro medio de rastreo de referencias a un paquete específico es el subcomando *search*. Esta función localizará todos los paquetes y dependencias relacionadas con el término de búsqueda, seguido de una breve descripción. Similar a apt-cache en sistemas Debian, *search* puede ser útil para encontrar paquetes cuando carecemos de un nombre exacto o estamos razonablemente seguros de que una función debe estar disponible en algún sitio.

Todos estos subcomandos de información ofrecen frecuentemente docenas, incluso cientos de líneas de salida. Por esta razón, debemos considerar redirigirlos a través del comando *less* de manera que podamos desplazarnos a nuestro placer. Por ejemplo, con *yum list obsoletes | less* podemos mirar una lista de los paquetes instalados que se han vuelto obsoletos por paquetes en los repositorios.

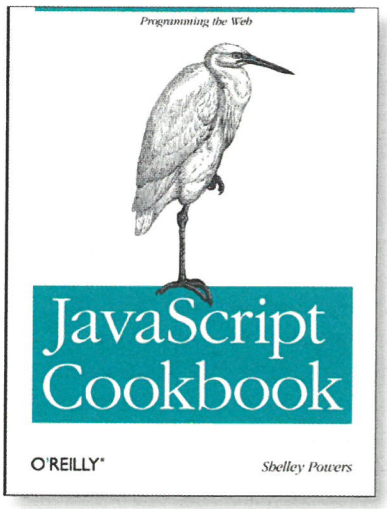
Subcomandos de Mantenimiento

Los subcomandos de Yum incluyen también una serie de utilidades para ayudarnos a mantener y resolver problemas de nuestro sistema. Por ejemplo, *yum makecache* descarga la información para todos los paquetes en todos los repositorios habilitados, los cuales podemos usar si la información está corrupta o si hemos cambiado repositorios recientemente. Asimismo, en el momento excepcional en el que emergen de repente los problemas de dependencias, *yum resolvedep* puede decirnos qué paquetes proporcionan una dependencia faltante.

Una herramienta de mantenimiento particularmente potente es *clean*, que, al igual que *list*, está completada por una descripción de la fuente de información que deseamos eliminar. Sin embargo, con la excepción del comando *yum clean packages*, que elimina paquetes que fueron descargados pero no instalados, el uso de *clean* es un acto de desesperación.

Ejecutar *clean* seguido de cualquier otra opción – *expire-cache*, *headers*, *meta-data*, *dbcache* o *all* – elimina información que Yum necesita para operar. La próxima vez que iniciemos Yum después de ejecutar *clean* con estas terminaciones, Yum volverá a crear lo que se borró, aunque podría llevarle hasta 20 minutos hacerlo, dependiendo de la máquina. Por esta razón, sólo deberíamos ejecutar el subcomando *clean* cuando estamos teniendo problemas con Yum y han fallado todos los demás medios de resolución de problemas. A diferencia de *clean* y *autoclean* de apt-get, *clean* de Yum no es para un mantenimiento de rutina, sino para problemas serios, y sólo nos causaremos molestias a nosotros mismos si lo ejecutamos de manera informal.

Programming the Web with JavaScript



JavaScript Cookbook
ISBN: 9780596806132
40 €

Covering both ECMAScript 5 and HTML5, JavaScript Cookbook helps you take advantage of the latest web features, including HTML5's persistent storage mechanisms and drawing canvas. You'll find solutions for integrating these features with JavaScript into UIs that people will enjoy using. The recipes in this book not only help you get things done, they'll also help you develop applications that work reliably in every browser.

O'REILLY®
www.oreilly.com

©2010 O'Reilly Media, Inc. O'Reilly logo is a registered trademark of O'Reilly Media, Inc.

Opciones

La mayoría de las veces podemos usar Yum sin opciones. Algunas, como *--obsoletes*, suministran información útil para ayudarnos a administrar la instalación de software. Una gran mayoría de veces, sin embargo, habilita o deshabilita información para varios propósitos.

Algunas opciones, tales como *-d* y *-e*, las cuales establecen informes de nivel de error y depurado, son en gran parte para desarrolladores. Lo mismo ocurre con *-v* o *--verbose*, dos opciones equivalentes que aumentan la cantidad de información que Yum suministra mientras se ejecuta.

Otras opciones son para usuarios que desean utilizar Yum con un mínimo de escándalo, como *--quiet*, que hace que Yum se ejecute sin informar de lo que está haciendo. Un compañero frecuente es *-y*, el cual supone que las respuestas a todas las preguntas es "Sí" – incluyendo la pregunta de si deseamos continuar después de que Yum finalice sus cálculos iniciales. De la misma forma, *--nogpgcheck* deshabilita la verificación del paquete.

Tales opciones también nos ahorran tiempo y molestias. Sin embargo, sugiero que se eviten según el principio general de que abandonar el control es raras veces una buena idea. Algo tan simple como un error de mecanografía podría hacer que Yum comenzara una serie de acciones que podrían destrozar nuestro sistema – o al menos que precisara algunas reparaciones.

Otras opciones es menos probable que causen problemas. La pareja *--enable-repo=* y *disable-repo=* dan a Yum el equivalente del direccionamiento de apt-get y especifican qué repositorios utilizar. También podemos usar *--exclude=* para evitar paquetes que podrían causar un conflicto de instalación desde cualquier fuente.

Otra opción que tampoco puede evitarnos problemas es *--skip-broken*. Si la usamos después de que Yum nos informe sobre la pérdida de una dependencia, debería permitirnos resolver la dificultad. En algunos casos, el paquete instalado con esta opción no funcionará, pero podemos asegurarnos de que no forma un cuello de botella que impide que Yum trabaje. Una vez que se han instalado, podemos eliminarlos con normalidad.

Plugins

```
firefox-3.5.2-2.fc11.i586 : Mozilla Firefox Web browser
Repo        : installed
Matched from:
Filename    : /usr/bin/firefox
Filename    : /usr/lib/firefox-3.5.2/firefox
```

Figura 4: Si nos preguntamos de dónde vienen los ficheros o aplicaciones, "yum list provides" nos puede facilitar esta información. Nos da ambas fuentes, las actuales y las posibles.

Los comandos de Yum y las opciones proporcionan toda la funcionalidad que necesitan la mayoría de los usuarios. Sin embargo, si estamos buscando algo extra, o si deseamos ver lo que puede ser el futuro de Yum, tomémonos un momento para examinar los plugins disponibles para nuestra distribución. Escritos en Python, los plugins de Yum se encuentran disponibles como paquetes separados en ella. Con cada plugin añadimos nuevas funcionalidades.

Por defecto, no se encuentran activados. Antes de poder usar ninguno de ellos, necesitamos abrir */etc/yum.conf* y añadir o editar la línea *plugins* en la sección principal del fichero para que diga *plugins=1*.

En Fedora y Red Hat, algunos de los plugins más comunes se encuentran disponibles en el paquete yum-utils. Actualmente también se encuentran disponibles otros 20. Sus funcionalidades cubren casi todo lo que podamos imaginar, siendo demasiados para describirlos aquí detalladamente.

Brevemente, sin embargo, algunos plugins útiles en Fedora 11 son:

- *yum-plugin-version-lock*: Evita que una versión particular del programa se sobrescriba.
- *yum-plugin-protects-packages*: Evita que sean eliminados paquetes designados (incluyendo Yum).
- *yum-plugin-allowdowngrades*: Nos permite bajar de versión un paquete – una operación que es a veces necesaria para compatibilidad entre paquetes.
- *yum-plugin-fastest-mirror*: Lista los mirrors más rápidos para los repositorios que pedimos.
- *yum-presto*: Ordena a Yum que busque DeltaRPMs en vez de RPMs. Los DeltaRPMs son paquetes que incluyen solamente cambios en un paquete, de este modo, usándolos, podemos instalar un paquete más rápidamente y con menos ancho de banda.
- *yum-plugin-security*: Añade opciones para limitar las actualizaciones a aquéllos que sean por seguridad.

Muchos plugins, si no todos, funcionan automáticamente, por lo que no incluyen ni páginas man ni de ayuda. Sin embargo, podemos ver los que se encuentran instalados en nuestro sistema con el comando *search yum*.

A veces podemos encontrar que un plugin, o quizás un conflicto entre dos o más plugins, impide que Yum funcione adecuadamente. Si esto ocurriera, podemos usar la opción *--disableplugin=[plugin name]* para ayudarnos a resolverlo deshabilitando un plugin específico. Si nos encontramos en apuros, podemos usar la opción *--noplugins* para ejecutar Yum sin plugins de manera que, con suerte, podamos recuperar el uso del programa.

Otros Administradores de Paquetes

Yum no es el único administrador de paquetes basado en RPM disponible. Mandriva usa una herramienta similar llamada urpmi y OpenSUSE utiliza Smart, un administrador que no sólo se parece a Yum, sino que podemos instalar desde los repositorios de Yum. Otros sistemas de administración de paquetes incluyen Portage and Conary de Gentoo, el cual incluye un control de versión que nos permite instalar diferentes versiones de los mismos paquetes.

Sin embargo, Yum es casi con toda seguridad el administrador de paquetes más comúnmente usado después de apt-get de Debian y Ubuntu. Además, como fue desarrollado años después que apt-get, sus desarrolladores tuvieron la oportunidad de aprender de apt-get y de mejorarlo.

A pesar de que probablemente no veremos mucho en Yum que no esté en apt-get y en sus utilidades relacionadas, sin embargo encontraremos una organización menos caótica. De hecho, si conocemos apt-get y estamos encontrándonos por primera vez con Yum, probablemente quedaremos sorprendidos por cómo está estructurado Yum y lo fácil que es aprenderlo si lo comparamos con apt-get. Como programa, Yum es una innovación inteligente – y como remedio para el demonio de las dependencias, una bendición. ■

Creación de imágenes ISO con dd y genisoimage

IMAGEN EN EL ESPEJO

123rf (Exclusivo), 123 RF

Siempre que tengamos que hacer copias de seguridad o crear CDs de arranque del sistema operativo, dd y genisoimage nos ayudarán con todo el trabajo, y además lo harán con mucho estilo.

POR HEIKE JURZIK

Hoy en día disponemos de muchas herramientas desde la línea de comandos que podemos usar para grabar datos en CDs y DVDs. Pero antes de poder copiar los datos a disco, necesitaremos crear lo que se conoce como imagen ISO. ISO (o ISO 9660, si queremos usar el nombre completo) es un sistema de ficheros estándar para la gestión de ficheros en CDs. Los programas *genisoimage* y *dd* nos ayudaran a crear estas imágenes ISO. La herramienta *genisoimage* soporta las extensiones Rockridge y Joliet (ver el cuadro titulado "Rockridge y Joliet"). Podemos utilizar *genisoimage* para hacer una copia de seguridad automática de nuestros datos, excluyendo algunos archivos individuales si fuera necesario. El programa *dd*, por otro lado, podría ser la última oportunidad de rescatar los datos de un disco duro dañado. *genisoimage* reemplaza a *mkisofs* en muchas de las distribuciones por razones de licencia; aún se puede usar el viejo comando *mkisofs*, ya que ahora éste es un enlace simbólico a *genisoimage*.

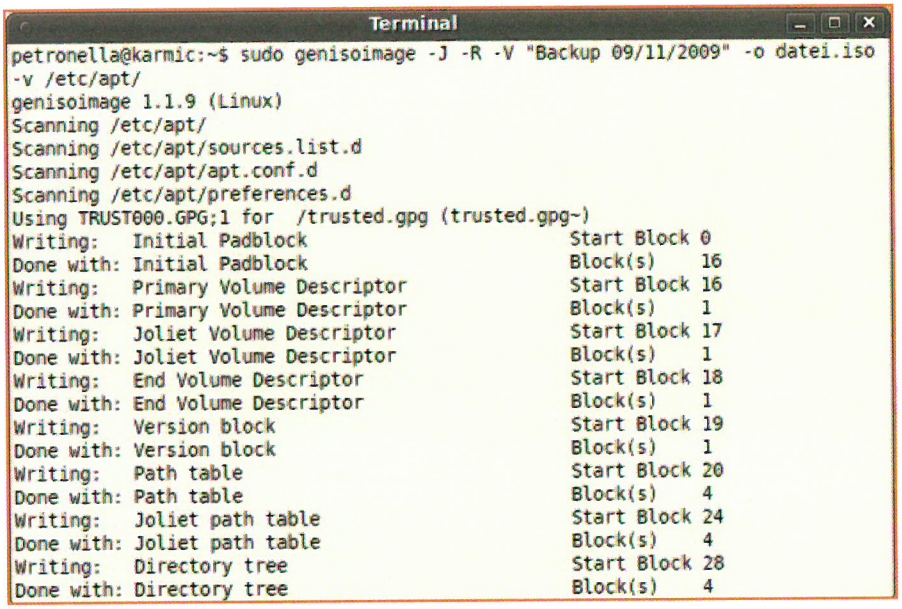


Figura 1: Genisoimage nos informa de lo que sucede en segundo plano.

Convertir y Copiar

La herramienta *dd* debería haberse llamado *cc*, de convertir y copiar. Pero como este nombre ya está siendo usado por el compilador de C, los desarrolladores decidieron usar la siguiente letra del alfabeto.

dd crea copias 1:1 de un medio, ya sean particiones del disco duro, CDs o DVDs. Es muy robusta en las operaciones de lectura y escritura en bloque. *dd* ni procesa ni interpreta los bloques de datos, por lo que el sistema de ficheros que exista por debajo no tiene importancia. De hecho, *dd* no se amilana ante discos duros con errores (ver la sección "Rescate en el último minuto"). La sintaxis básica del comando *dd* es:

```
dd if=source of=target
```

La opción *if* indica a *dd* dónde leer los datos de entrada, y la opción *of* dónde guardarlos.

Generalmente, las fuentes de entrada y destino suelen ser dispositivos, como particiones de disco duro o unidades de CD y DVD. También existe la alternativa de usar un fichero en vez de un dispositivo. Por ejemplo, para copiar la partición */dev/hda1* del disco duro bit a bit a */dev/hdb1*, deberíamos escribir:

```
dd if=/dev/hda1 of=/dev/hdb1
```

Podemos usar el comando *dd* para copiar CDs y DVDs directamente desde la línea de comandos. Por ejemplo, para crear una imagen ISO sólo hay que escribir:

```
$ dd if=/dev/hdc of=myimage.iso
1529961+0 records in
1529960+0 records in
783339520 bytes (783 MB) copied,
90.6944 s, 8.6 MB/s
```

Para realizar copias con *dd* no es necesario montar el dispositivo. Sólo hay que cambiar su nombre (*/dev/hdc*) por aquel que corresponda a nuestro sistema. La imagen ISO se escribe en el fichero *myimage.iso* situado en el directorio actual.

Glosario

ISO: Linux es capaz de trabajar con varios sistemas de ficheros, como ext2, ext3, ext4, ReiserFS o, incluso, FAT y NTFS. ISO 9660 (nombre completo del sistema de ficheros) es el sistema de ficheros estándar para la gestión de ficheros en CD-ROMs.

Opciones de Optimización

El programa *dd* ofrece varias opciones. Un parámetro usado para acelerar el proceso de copia es *bs* (de "*block size*" o tamaño de bloque). Por defecto, *dd* usa bloques de 512 bytes – es decir, lee 512 bytes de una sola vez y los escribe en el fichero de salida. Si escogemos bloques de mayor tamaño, podríamos acelerar la operación de copia. Por ejemplo:

```
dd if=/dev/hda1 of=/dev/hdb1 ➥
bs=2k
```

Este comando informa a *dd* que la partición debe ser copiada en bloques de 2KB (2048 bytes). Si el último bloque es menor que el tamaño de bloques indicado, *dd* no rellenará el espacio restante del bloque con ceros (no realizará *padding* sobre ese bloque):

```
$ dd if=/dev/hda1 ➥
of=/dev/hdb1 bs=6k
16059+1 records in
16059+1 records out
98670592 bytes (95 MB) copied
13.801482 s, 6.9 MB/s
```

La salida nos informa de que *dd* ha copiado 16059 bloques de 6144 bytes cada uno, más un bloque de tan sólo 4096 bytes.

Además del tamaño, podemos indicar el número de bloques que *dd* debe leer. Para copiar 40MB, sólo hay que escribir *bs = 1M count = 40*. El parámetro *count* especifica el número de bloques a leer. Esto es útil cuando queremos guardar el sector de arranque de un disco duro – sólo queremos copiar el primer bloque de 512 bytes:

```
dd if=/dev/hda of=bootsector ➥
bs=512 count=1
```

Rescate en el Último Minuto

La herramienta *dd* puede ser clave a la hora de enfrentarnos a la tarea de rescatar datos de un sistema de ficheros

Rockridge y Joliet

Las extensiones Rockridge proporcionan información tipo Unix (como puede ser 'dueño del fichero', grupo, privilegios y enlaces simbólicos) al sistema de ficheros ISO, para así evitar pérdida de datos cuando se copian desde un sistema Unix a un CD. De la misma forma, Rockridge permite el uso de nombres de ficheros largos. La extensión Joliet de Microsoft para el estándar ISO 9660 soporta también nombres de ficheros largos.

corrupto. Antes de arreglar el posible daño, lo primero sería hacer una copia de seguridad. Para hacerlo usaremos *dd* para crear una copia 1:1 de todo el sistema dañado, y luego utilizar esta copia en el proceso de recuperación de datos.

Puesto que *dd* excluye de manera predeterminada los sectores defectuosos durante el proceso de copia, necesitaremos activar la opción *conv=noerror,sync*,

```
dd bs=512 conv=noerror,sync ➥
if=/dev/hda of=/dev/hdb
```

que informa a *dd* para que pueda continuar el proceso de lectura y almacenado de datos, aún en el caso de que encuentre algunos sectores defectuosos. La etiqueta *noerror* informa a *dd* para que no pare al detectar el error, y *sync* rellena con ceros los sectores que no han podido ser leídos.

Imágenes ISO

La herramienta *genisoimage* solamente sirve para crear imágenes ISO, aunque por su condición de herramienta especializada lo hace de forma eficiente, proporcionando muchas y útiles características. La sintaxis básica es:

```
genisoimage parámetro ➥
-o myfichero.iso /dir/datos
```

El parámetro *-o* se usa para indicar el nombre del fichero de destino. Luego sigue el directorio que contiene los datos que deseamos almacenar en la imagen. Podemos decir a *genisoimage* que queremos activar la extensión Rockridge usando el parámetro opcional *-r*. Para establecer privilegios y propiedad sobre los ficheros alojados dentro de la ISO, deberemos usar la opción *-R*. Las extensiones Joliet se activan por medio del parámetro *-J*:

```
genisoimage -J -R ➥
-o myfichero.iso ➥
/dir/datos
```

La opción *-V* nos permite indicar el nombre (o identificador de volumen) para el CD/DVD. Si el nombre incluye espacios, caracteres especiales o ambos, no debemos olvidar usar el entrecomillado:

```
genisoimage -V "Backup" ...
```

Para obtener una salida más detallada, podemos usar la opción *-v* (de "verbose"). La opción opuesta sería *-quiet*. Si preferimos evitar mensajes de estado en la ventana del terminal, pero no deseamos perder esa información, deberemos redireccionar la salida de *genisoimage* a un fichero de log:

```
genisoimage ... -log-file ➥
log.txt
```

Idóneo para Backups

genisoimage tiene varias opciones muy prácticas para crear copias de seguridad. Por ejemplo, la opción *-m* nos permite excluir ficheros de nuestra imagen. Los nombres de los ficheros deben seguir a esta opción. *genisoimage* puede incluso aceptar en los parámetros de eliminación comodines y múltiples nombres de ficheros. Por ejemplo, para excluir todos los documentos HTML de nuestra imagen ISO, deberemos escribir:

```
genisoimage ... -m *.html ➥
-m *.HTML -o backup.iso ➥
/home/huhn
```

La opción *-x* nos permite excluir directorios completos; también soporta múltiples argumentos, como vemos a continuación:

```
genisoimage ... -x /tmp ➥
-x /var -o backup.iso /
```

Cuando usemos estos parámetros, deberemos asegurarnos de evitar los comodines a la hora de indicar los ficheros que deseamos escribir en la imagen. Por ejemplo, el siguiente comando:

```
genisoimage ... -m *.html ➥
-m *.HTML -o backup.iso *
```

le indica a la shell que resuelva el último comodín, añadiendo todos los ficheros a nuestra imagen.

Si deseamos excluir ficheros que contienen las típicas extensiones de backup, que suelen terminar en ~, # o .bak, sólo hay que usar la opción *-no-bak*.

¡Arranca!

Para crear un disco de arranque deberemos añadir el cargador Isolinux [1], el cual trabaja mano a mano con *genisoimage*:

```
genisoimage -J -R -o bootcd.iso ➥
-b isolinux/isolinux.bin ➥
-c isolinux/boot.cat ➥
-no-emul-boot ➥
-boot-load-size 4 ➥
-boot-info-table ➥
/folder/data
```

Como puede observarse, se han usado algunas opciones nuevas. El parámetro *-b* debe ir seguido por el nombre de la imagen de arranque, y con *-c* podemos establecer el catálogo de arranque.

El parámetro *-no-emul-boot* informa al programa que la imagen que se usa para el arranque no proporciona ningún tipo de emulación de *floppy* durante el proceso de carga. La opción *-boot-load-size 4* se utiliza para establecer que la BIOS debe cargar 4 sectores de 512 bytes del fichero de arranque. Finalmente, la opción *-boot-info-table* indica que se ha de crear la tabla que contiene la estructura del CD, y que es leída durante el proceso de arranque. Toda esta información debe estar alojada en el directorio *isolinux* bajo */folder/data*.

Probando la Imagen

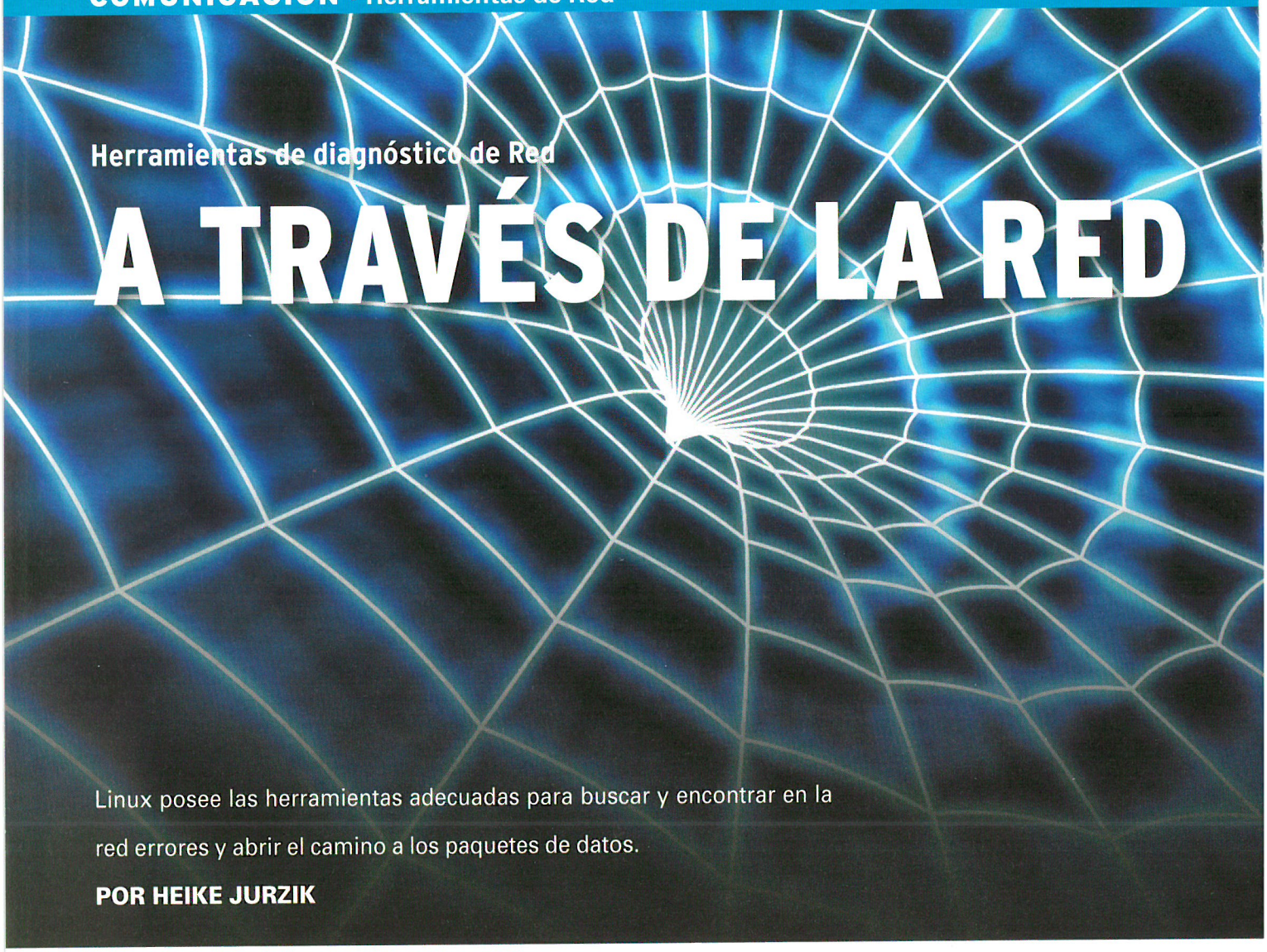
La herramienta *mount* nos permite, de forma práctica, comprobar las imágenes ISO antes de proceder al copiado en el CD. Para probar la imagen, sólo deberemos montarla en nuestro sistema de ficheros especificando la opción *-o loop*:

```
mount -o loop myfichero.iso ➥
/mnt/tmp
```

El punto de montaje debe existir previamente, y no son necesarios permisos de administrador para poder ejecutar el comando. Tras completar el test, podremos desmontar la imagen ISO ejecutando *umount /tmp/mnt*. ■

RECURSOS

[1] Isolinux: *http://syslinux.zytor.com/iso.php*



Herramientas de diagnóstico de Red

A TRAVÉS DE LA RED

Linux posee las herramientas adecuadas para buscar y encontrar en la red errores y abrir el camino a los paquetes de datos.

POR HEIKE JURZIK

Obtener ayuda puede salir muy caro si la conexión a Internet falla inesperadamente o cuando no se puede tener acceso a los ordenadores de la red local. Varias herramientas de la línea de comandos pueden ayudar a investigar conflictos en la red. *ping* y *traceroute* comprueban la disponibilidad de otros ordenadores; puede contactarse con los servidores con *host* y *dig*; *netstart* ayuda a descubrir qué está ocurriendo en el sistema.

¿Quién Soy Yo?

La herramienta *ifconfig* permite comprobar la propia interfaz de red del ordenador personal. Esta práctica utilidad no sólo presenta información de la configuración actual, sino que también ayuda a configurar la interfaz. Para presentar las configuraciones actuales se ejecuta */sbin/ifconfig* sin ningún parámetro. Como el programa reside en */sbin*, que habitualmente no se encuentra en *$PATH* del usuario, será necesario especificar la ruta completa.

El Listado 1 muestra el comando y la salida para cada interfaz en secciones separadas. En este ejemplo el ordenador posee una tarjeta de red (*eth0*) con una dirección IP de (*inet addr*) *10.195.34.14*. Pueden verse los detalles de la dirección de emisión del programa (*Bcast*), y la máscara de red (*Mask*). La etiqueta *UP* muestra el número de paquetes recibidos, y *TX* el número de paquetes transmitidos.

Adicionalmente, se encuentra configurada la interfaz loopback (*lo*), que es la que da acceso interno a los usuarios a la máquina a través de la dirección IP *127.0.0.1* y el nombre de *localhost*. La tercera interfaz de esta lista es una conexión DSL, *ppp0*; el ordenador tiene una dirección en Internet de *11.22.33.44*.

¿Quiénes son los Demás?

Gracias a una práctica invención llamada *Domain Name Service*, nadie está obligado a recordar las complejas direcciones IP. En su lugar pueden usarse nombres de dominios para contactar con los ordenadores. *dig* o *host* en la línea de comandos permiten comprobar si la configuración DNS está funcionando adecuadamente. Ambas herramientas esperan bien un nombre de dominio o bien una dirección IP como argumento, y la resolución del nombre funciona en ambas direcciones.

El Listado 2 es un ejemplo de una petición *dig*. Además de *QUESTION SECTION*, *dig* también presenta una *ANSWER SECTION*. La dirección IP para el nombre de dominio pasó a *dig* seguido de A. Para resolverlo en la

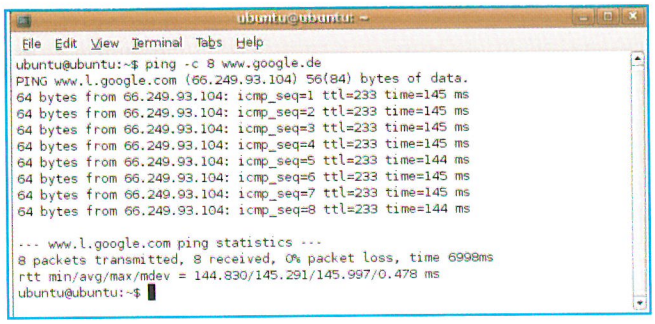


Figura 1: Ping permite comprobar si la máquina está accesible.

Listado 1: Salida ifconfig

```
$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:30:48:70:4B:40
          inet addr:10.195.34.14   Bcast:10.195.34.255
   Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST   MTU:1500  Metric:1
          (...)
          RX packets:1162180567 errors:0 dropped:449 overruns:0   frame:0
          TX packets:2046191782 errors:0 dropped:0 overruns:0   carrier:0
          (...)
lo        Link encap:Local Loopback
          inet addr:127.0.0.1   Mask:255.0.0.0
          (...)
ppp0      Link encap:Point-Point Protocol
          inet addr:11.22.33.44   P-t-P:11.22.33.55 Mask:255.255.255.0
          (...)
```

dirección opuesta, es decir, desde una dirección IP a un nombre de dominio, es necesario especificar la opción *-x*.

La herramienta *host* no necesita un parámetro para especificar la dirección a resolver, aunque tampoco ofrece tanta información como *dig*. Acepta IPs y nombres de dominios sin protestar. Hay un ejemplo en el Listado 3: algunos nombres de dominio comparten a menudo la misma dirección IP, cuando un servidor web hospeda diferentes páginas, por ejemplo. Al mismo tiempo, un nombre de dominio puede resolver a múltiples direcciones IP: raramente usa sitios webs como Google o eBay, por razones de rendimiento usa múltiples servidores.

¿Bien Conectado?

Si tu ordenador no se encuentra conectado directamente a Internet, sino que se usa un router, puede ejecutarse el comando *route* para obtener la tabla routing. Es preciso introducir de nuevo la ruta completa para este comando. El parámetro *-n* configura adicionalmente las salidas de las direcciones IP en lugar de nombres de hosts.

La salida desde este comando es una tabla con el destino como primera columna (*Destination*). La última columna contiene la interfaz a través de la cual se enviarán y recibirán los paquetes. En nuestro ejemplo (Listado 4), todos los paquetes enviados a las direcciones IP que comienzan con *10.195.34* usan la interfaz de red *eth0*. La conexión a Internet se muestra en la segunda línea; si la dirección destino no comienza con *10.195.34*, el paquete será enviado primero a *10.195.34.5* (el router) que se ocupará de remitirlo a su destino.

Ping-Pong

El comando *ping*, común a muchos sistemas operativos, permite comprobar si una máquina se encuentra activa. El programa espera una dirección IP o un nombre de host como entrada, y mide la respuesta del destino. Para comprobar si está funcionando la conexión a Internet puede hacerse ping a una máquina fuera de la propia red, tal como *ping -c 8 www.google.com*.

El programa envía paquetes ICMP al ordenador especificado, el cual, habitualmente, responderá con paquetes ICMP. Si no se especifica ningún parámetro adicional, la prueba continuará hasta que deje de hacerse ping pulsando el atajo de teclado [Ctrl] + [C]. La opción *-c número* permite restringir el número de paquetes a intercambiar. Después de completarse el intercambio de paquetes, ping presenta estadísticas con los tiempos de viaje, que adicionalmente dirán el número de paquetes que se han perdido (Figura 1).

ping es una muy buena manera de aislar algunos errores de comandos:

- Si se arranca el programa y no ocurre nada, pero se ve un mensaje de error tal como *ping: unknown host www.google.de* poco tiempo después, probablemente sea porque hay un problema DNS. En este caso hay que intentar hacer ping a una dirección IP para comprobar las configuraciones de red.
- Si todo es un poco lento y *ping* no proporciona ninguna salida primero, aunque

Listado 2: Ejemplo de dig

```
$ dig www.linux-magazine.es

; <<>> DiG 9.3.1 <<>> www.linux-magazine.es
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29166
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.linux-magazine.es.          IN      A

;; ANSWER SECTION:
www.linux-magazine.es.  85779   IN      A       212.227.96.14

;; AUTHORITY SECTION:
linux-magazine.es.      82832   IN      NS      ns2.m-online.net.
linux-magazine.es.      82832   IN      NS      ns1.m-online.net.

;; ADDITIONAL SECTION:
ns1.m-online.net.       68887   IN      A       212.18.0.8
ns2.m-online.net.       68884   IN      A       212.18.3.8

;; Query time: 2409 msec
;; SERVER: 194.25.0.69#53(194.25.0.69)
;; WHEN: Mon Jan  1 21:18:31 2007
;; MSG SIZE  rcvd: 135
```

Listado 3: Nombre de Dominio con Múltiples IPs

```
$ host www.google.com
www.google.com is an alias for www.l.google.com.
www.l.google.com has address 66.102.9.104
www.l.google.com has address 66.102.9.147
```

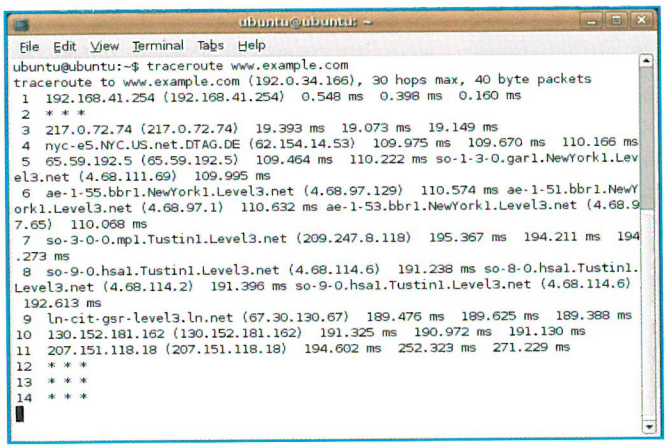


Figura 2: La herramienta traceroute muestra las estaciones intermedias en ruta desde tu ordenador a la estación destino.

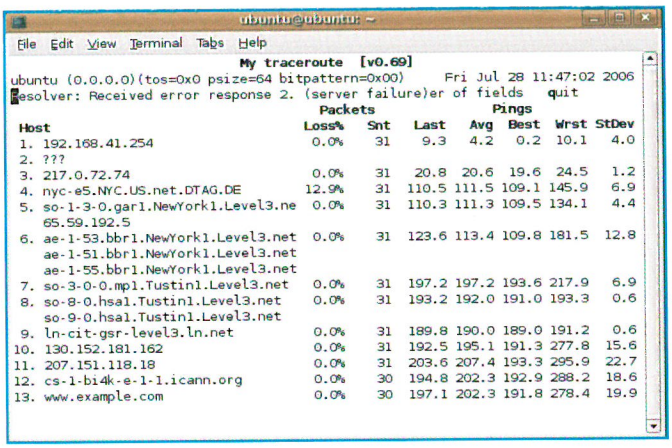


Figura 3: mtr combina la funcionalidad de las utilidades ping y traceroute.

luego muestra la salida normal, el primer servidor de nombres puede que no se encuentre disponible o bien podría tener algún otro problema. En este caso, el segundo servidor de nombres de la configuración entra en funcionamiento después de una corta demora.

- Si no ve ninguna respuesta, el cortafuegos puede estar bloqueando los paquetes ICMP por el lado objetivo.
- *packet loss* no significa automáticamente que se tenga un problema, el objetivo del router podría estar ocupado.

La Ruta Adecuada

Aunque ping nos dice si se puede llegar hasta un servidor, no ofrece información acerca de la ruta que siguen los paquetes a través de la red. Para eso se necesita la herramienta *traceroute*. De nuevo se le pasa la dirección IP o el nombre de host a la herramienta (Figura 2).

Al igual que ping, *traceroute* envía paquetes traza a través de la red. Los paquetes tienen un tipo de "fecha de caducidad" (TTL = Time to Live, o Tiempo de vida). Cuando un paquete alcanza un host, éste envía un mensaje de error a la máquina fuente y rechaza el paquete. Si el paquete es válido, decrementa el TTL y envía el paquete a la siguiente estación en ruta.

Los mensajes de error rastrean la ruta. *traceroute* incrementa el TTL en cada etapa para permitir que los paquetes viajen a una estación más. Un paquete con un TTL de 2 alcanzará la segunda dirección intermedia, TTL 3 alcanzará la tercera, y así sucesivamente. *traceroute* repite esta transmisión tres veces para cada máquina, y espera tres segundos para que cada máquina individualmente responda.

traceroute puede tomar diferentes rutas. Si se envía a la misma máquina dos veces y se ven diferentes direcciones intermedias, no hay que preocuparse. Si se ven asteriscos como en la Figura 2, probablemente se alcanzará un cortafuego en ruta, aunque los paquetes simplemente pueden desaparecer.

traceroute usa por defecto paquetes UDP, aunque especificando la opción *-I* podrán usarse paquetes ICMP. Si la salida contiene una cadena de asteriscos inusualmente larga, puede suponerse que la herramienta ha caído a merced de todos esos cortafuegos. En este caso, debería probarse con *tcptraceroute*, un programa que transmite paquetes TCP configurados para usar el puerto 80, el puerto al que escuchan la mayoría de los servidores web. Si se sabe que el destino no ejecuta un servicio web, hay que especificar el puerto. Para contactar con una máquina a través del puerto 22 (SSH), se escribe lo siguiente:

```
$ tcptraceroute target 22
```

Otras Herramientas

mtr es una ingeniosa combinación de ping y traceroute. Es preciso especificar la máquina destino cuando se arranca. La salida resultante muestra la ruta tomada por los paquetes, presentándose las estaciones individuales y continuando con ping hasta que se abandone cuando se pulse [Q]. Como alternativa, puede especificarse un número determinado de paquetes; por ejemplo, *mtr -c 12 targets* abandona mtr después de enviar una docena de paquetes al destino.

netstat es otro programa que permite descubrir errores en la configuración de red. Sin ningún parámetro, vuelve al estado de sockets abiertos.

El parámetro *-e* ofrece información adicional sobre el UID, *-p* suministra información sobre el PID y el nombre del programa que abrió el socket. *-a* muestra todos los servicios, bien servicios activos o sockets de servidores escuchando conexiones. Si solamente se está interesado en conexiones TCP, puede especificarse adicionalmente la opción *-t*. El ejemplo del Listado 5 muestra la ejecución de los servicios FTP y SMTP. Un usuario ha abierto una conexión FTP a la máquina y otro está navegando por la web. ■

Listado 4: Tabla de Enrutado

```
$ /sbin/route -n
Kernel IP routing table
Destination  Gateway     Genmask         Flags Metric Ref Use Iface
10.195.34.0  0.0.0.0     255.255.255.0 U     0      0      0 eth0
0.0.0.0      10.195.34.5 0.0.0.0       UG    0      0      0 eth0
```

Listado 5: Salida de netstat

```
$ netstat -atp | less
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State (...)
tcp        0      0 *:ftp           *               LISTEN -
tcp        0      0 *:smtp          *               LISTEN - (...)
```

Conexiones seguras mediante SSH

ZAPADOR DE TÚNELES

Siempre que necesitemos tunelizar conexiones para el envío de datos cifrados entre varios PCs o aplicaciones gráficas por medio de la red, la gran herramienta SSH nos proporcionará toda la funcionalidad que precisamos .

POR JÖRG HARMUTH

Telnet siempre ha sido la herramienta de referencia a la hora de necesitar conexiones remotas por medio del terminal. Pero este viejo dinosaurio de las redes de comunicaciones tiene una desventaja importante: todos los datos se transmiten en claro a través de la red. Si un atacante pudiera fisgonear los datos de la conexión, en poco tiempo vería circular la contraseña de administración del servidor. Hay que admitir que quizás no sea tan fácil, pero el peligro sigue estando de todas formas. Por esto, ya todas las distribuciones de Linux instalan SSH (Secure Shell) como alternativa segura.

Los ficheros de configuración de SSH se suelen encontrar siempre en */etc/ssh*. Existe un fichero de configuración para el servidor (*sshd_config*) y otro para el cliente (*ssh_config*). Ambos contienen un buen número de opciones, explicadas todas ellas en las páginas del manual. Normalmente, el usuario no debería tener que modificar estos ficheros. Los valores establecidos para la versión 11.3 de openSUSE son suficientemente cómodas y seguras como para necesitar algún tipo de ajuste.

Cómo Funciona

SSH utiliza una arquitectura de tipo cliente/servidor construida sobre el protocolo TCP/IP. Una vez lanzado el servicio SSH (*sshd*), éste se queda en segundo plano y escuchando por el puerto TCP 22. El cliente, por tanto, debe hacer uso de este mismo puerto para el establecimiento de las conexiones con el servidor.

Durante el establecimiento de la conexión, suceden varias cosas en segundo plano. Primero, el cliente y el servidor negocian la versión del protocolo SSH que se usará para el resto de la conexión. Actualmente hay disponibles dos versiones: SSH1 y SSH2, pero la versión SSH2 se encuentra hoy en día mucho más difundida debido a sus mejoras en cuanto a seguridad. Algunos de los detalles se recogen en el cuadro "Versiones del Protocolo SSH".

Segundo, el servidor y el cliente negocian el algoritmo y la contraseña que se usará para la transferencia de datos. Se genera una clave que será usada de forma exclusiva para la sesión de comunicación en curso, y ambos extremos la destruirán una vez termine esta conexión. Para sesiones de larga duración, la clave de cifrado cambiará a intervalos regulares (establecidos en una hora por defecto).

```
debian:~# ssh sector
The authenticity of host 'sector (192.168.10.100)' can't be established.
RSA key fingerprint is 81:00:6e:dc:49:e1:5b:1d:76:86:8c:a4:55:91:0d:29.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added 'sector,192.168.10.100' (RSA) to the list of known host
Password:
Last login: Tue Sep 27 14:45:53 2005 from 192.168.10.254
Loading /usr/share/keymaps/i386/qwertz/de-latin1-nodeadkeys.kmap.gz
SECTOR:~$
```

Figura 1: Durante el login inicial, SSH importa la clave asociada a la máquina remota.

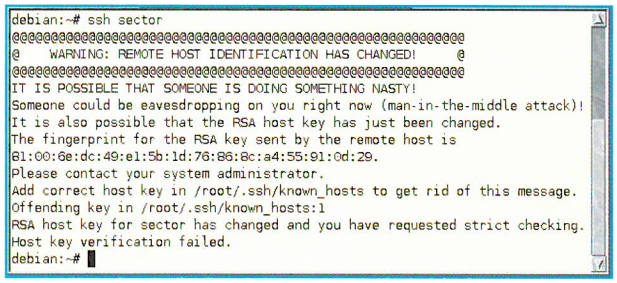


Figura 2: Si la clave asociada a la máquina cambia, el cliente SSH impedirá la conexión.

Login Inicial

La forma más simple de identificarse en el sistema es mediante el clásico sistema usuario/contraseña. El cliente SSH usa por defecto el nombre de usuario de la cuenta actual si no se le indica otro diferente. Durante el primer login, el cliente desconoce la clave pública de la máquina servidora, por lo que preguntará al usuario confirmación sobre el establecimiento de la conexión con la máquina remota. Tras esa confirmación, el cliente generará una huella resumen asociada a la clave del servidor (Figura 1).

Un buen consejo sería comprobar que la huella generada corresponde realmente al servidor al que nos queremos conectar. Para ello deberemos contactar con el administrador de sistemas de la máquina remota. Esta comprobación podría evitar ataques tipo '*man-in-the-middle*'. En este tipo de ataques, se suele redireccionar el tráfico de la red a una máquina controlada por el hacker. De esta forma se podría manipular el proceso de login. Si el cliente aceptara la huella del servidor como válida e introdujera la contraseña, el atacante terminaría por conocer nuestra palabra de paso. Se recomienda, por tanto, mucha precaución a la hora de aceptar huellas de servidores remotos.

Si en algún momento cambiara la clave del servidor, el cliente comprobaría que la huella generada es diferente a la que tiene almacenada, y por seguridad rechazará cualquier intento de conexión a ese servidor. La Figura 2 muestra la salida del cliente SSH. Para poder volver a conectarnos al servidor, deberemos eliminar la huella afectada del fichero *$HOME/.ssh/known_hosts* y aceptar la nueva clave una vez contactado el administrador de la máquina remota. Para que SSH se comporte de esta forma, es necesario tener establecida la variable *StrictHostKeyChecking* en el fichero *ssh_config*.

Cuando queramos conectarnos con un usuario diferente al de la cuenta actual deberemos usar la opción *-l username*. Por ejemplo, el comando *ssh -l tuppes sector* nos permitirá hacer login a la máquina remota como el usuario *tuppes*. SSH también acepta la sintaxis *ssh tuppes@sector*. Para ejecutar un comando en la máquina remota, es suficiente pasarlo como parámetro al final de la línea de comandos (Listado 1).

Si está cansado de tener que escribir siempre la contraseña en cada una de las conexiones realizadas, puede utilizar un sistema de autenticación basado en clave pública. El sistema de cifrado que hay bajo este método se encuentra también en otras aplicaciones como GnuPG. Para poder utilizarlo hay que usar el programa *ssh-keygen* para generar el par de claves pública y privada: *ssh-keygen -b 1024 -t rsa*. El programa avisa una vez termine de generar el par de claves pública y privada necesarias para la codificación RSA. Cuando el sistema nos pida que introduzcamos una contraseña, deberemos pulsar dos veces el retorno de carro. Tras esto, el programa nos informará de dónde ha almacenado los datos y nos mostrará la huella generada a partir de la nueva clave.

En el ejemplo anterior, el programa genera un par de claves RSA (opción *-t rsa*) con longitud de 1024 bits (opción *-b 1024*). La clave RSA es compatible con ambas versiones del protocolo. Por razones de seguridad, la longitud de la clave no debería ser menor de 1024 bits. Para estar más tranquilo, en caso de necesidad se deberían usar claves de hasta 2048 bits. Basándose en el estado del arte actual, estas claves se pueden considerar seguras hasta el año 2020. Además, el uso de claves largas no ralentiza en ningún momento la velocidad de transferencia de los datos, ya que éstas no son usadas para el cifrado de esos datos. El siguiente paso consiste en copiar la clave pública al fichero *$HOME/.ssh/authorized_keys* de la máquina remota usando, por ejemplo, un *pendrive* USB:

```
mount /dev/sda1 /media/usb
cat /media/usb/id.rsa.pub >>
 $HOME/.ssh/authorized_keys
umount /media/usb
```

Se debería evitar transferir la clave usando algún medio no seguro como el correo o FTP. La Figura 3 muestra el, ahora poco espectacular, proceso de login usando este nuevo mecanismo.

Otra buena costumbre es la de proteger las claves con contraseña. Cada vez que hagamos un uso interactivo de las claves generadas, el sistema nos pedirá la contraseña asociada; de otro modo, cualquier persona con acceso físico al ordenador podría usar las claves para entrar a la máquina remota. Los mecanismos de acceso basados en claves no protegidas con contraseña (como en este ejemplo) suelen usarse en tareas que deben ejecutarse de forma automática. Por ejemplo, si cada tarde se realizan copias de seguridad de los datos y se quiere que de forma automática estos datos se copien a la máquina remota, las claves sin contraseña pueden ser de utilidad. Si la clave se encuentra protegida por contraseña, se nos pedirá cada vez que se vayan a copiar datos. Con lo que el proceso ya no sería ahora tan automático.

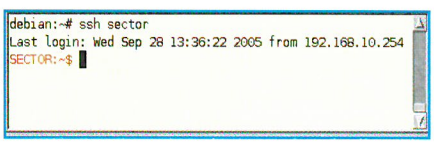


Figura 3: La autenticación por medio de clave pública simplifica el proceso de login al eliminar el paso de introducción de la contraseña.

Listado 1: Ejecución de Comandos en el Ordenador Remoto

```
01 jha@scotti:~$ ssh sector "ls  l"
02 Password:
03 insgesamt 52
04 Drwxr xr x 3 tuppes users 4096 2005 08 26 12:38 .
05 Drwxr xr x 16 root root 4096 2005 09 07 13:47 ..
06  rw rw r   1 tuppes users 266 2005 04 12 12:00 .alias
```

```
debian:~# netstat -tlpn | grep 23 | grep ssh
tcp        0      0 127.0.0.1:23            0.0.0.0:*               LISTEN      3311/ssh
tcp6       0      0 :::22                   :::*                    LISTEN      2364/sshd
tcp6       0      0 ::1:23                  :::*                    LISTEN      3311/ssh
debian:~# netstat -tpn | grep ssh
tcp        0      0 192.168.10.254:32790    192.168.10.100:22       ESTABLISHED3311/ssh
debian:~#
```

Figura 4: El comando netstat nos muestra el túnel SSH.

Extras de Utilidad

El paquete SSH incluye dos programitas interesantes: Copia Segura (*scp*) y FTP Seguro (*sftp*). Tal y como indican los nombres, estos programas son usados para copiar y transferir ficheros por FTP vía SSH. La sintaxis de ambos programas son similares.

Por ejemplo, el siguiente comando copia un fichero llamado *test.txt* desde el directorio de usuario de la máquina remota al directorio de trabajo actual:

```
scp OrdenadorRemoto:test.txt .
```

Dependiendo del sistema de autenticación, podría ser necesario introducir la contraseña o no. Los dos puntos son siempre necesarios. Separan el nombre de la máquina de la ruta donde se encuentra el fichero. Se debe indicar también la ruta local donde se desea copiar el fichero (el punto usado representa el directorio de trabajo actual). Si se desean copiar múltiples ficheros, sólo hay que indicar la lista de sus nombres separados por espacio:

```
scp OrdenadorRemotoA:test1.txt ➥
  OrdenadorRemotoB:test2.txt .
```

Si usamos autenticación por contraseña, el cliente SSH nos pedirá una contraseña por cada fichero que se copie. Pero si usamos el método de clave pública explicado anteriormente, no habrá necesidad de escribir contraseña alguna. El comando *scp OrdenadorRemotoA:test1.txt OrdenadorRemotoB:* copiará el fichero de la máquina remota A a la máquina remota B. Cuando se desean realizar operaciones de copia usando un usuario diferente, se puede usar la sintaxis:

```
scp tuppes@OrdenadorRemoto:➥
files/test.txt .
```

A diferencia de SSH, con *scp* no podemos usar la opción *-l username*. Si se está copiando en la dirección inversa – desde la máquina local a la remota – el procedimiento es igual de simple:

```
scp ./test.txt ➥
  tuppes@OrdenadorRemoto:files/
```

El comando *scp* copia el fichero *test.txt* desde el directorio de trabajo actual al directorio */home/tuppes/files* de la máquina remota.

El comando *sftp* tiene un conjunto de opciones similares al de *scp*, pero a diferencia de éste puede funcionar en dos modos distintos: en un modo interactivo, como el que es costumbre en FTP, y en modo *batch*. Si queremos usar *sftp* para transferir el fichero de ejemplo en este último modo, deberíamos escribir:

```
sftp OrdenadorRemoto:test.txt .
```

Si escribimos *sftp OrdenadorRemoto: test.txt remote_ test.txt*, el programa renombrará la copia local del fichero a *remote_test.txt*. Simplemente escribiendo *sftp OrdenadorRemoto* abriremos una sesión FTP interactiva y cifrada a la máquina remota. El servidor aceptará comandos FTP como *GET* o *PUT* durante la sesión.

Construyendo Túneles

SSH puede usarse para encapsular dentro de él otros protocolos diferentes. Por ejemplo, podemos ejecutar el protocolo Telnet sobre una conexión SSH cifrada de forma transparente al usuario. El término técnico usado para designar el encapsulado de un protocolo dentro de otro se conoce como tunelización.

Si queremos utilizar el túnel, los programas deben estar ejecutándose sobre la misma máquina. Si se desea que otra máquina diferente pueda utilizar el túnel, se debería indicar la opción *-o GatewayPorts = yes* al establecerlo. Una alternativa sería la de establecer la opción dentro del fichero de configuración *ssh_config*.

Esta configuración nos aproxima a los modelos VPN (Red Privada Virtual), aunque es más sencilla de implementar. La variante basada en SSH tiene un problema, y es que ésta sólo nos permite el redireccionamiento de un solo puerto TCP. Es decir, necesitaremos un túnel SSH por cada puerto que deseamos redireccionar. Por tanto, si deseamos cifrar toda comunicación entre dos máquinas, la solución VPN puede llegar a ser más adecuada.

Cualquier usuario puede establecer un túnel, aunque como es lógico, todos los puertos por debajo de 1024 quedan reservados para el usuario *root*. Para abrir un túnel a una máquina remota para encapsular el protocolo de Telnet (puerto 23), habría que escribir lo siguiente:

```
ssh -c blowfish -L ➥
  23:RemoteBox:23 RemoteBox
```

Este comando hace uso de la opción *-L* para abrir un túnel desde el puerto 23 de la máquina local (el primer 23) al puerto 23 de la máquina remota. Se utiliza el método de cifrado Blowfish por su velocidad. Si escribimos dos nombres diferentes de máquinas remotas, podremos sacar partido de otra de las características de SSH: la capacidad de construir túneles que hagan uso de una máquina intermedia entre la remota y la local. El siguiente comando:

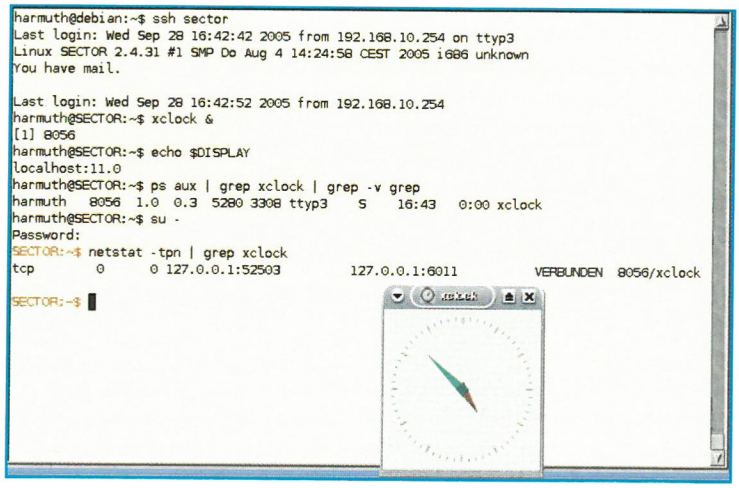


Figura 5: Conexión X11 tunelizada - el comando Xclock se está ejecutando en la máquina remota.

```
ssh -L 23:192.168.1.1:23 ➥
192.168.20.5
```

crea un túnel en la máquina local, y lo redirige por medio de la máquina 192.168.1.1 a su extremo final (192.168.20.5). La sintaxis general para la apertura de túneles desde la máquina local a la remota es, por tanto: *ssh -L PuertoLocal:OrdenadorRemotoA:RemotePort OrdenadorRemotoB*. Para túneles directos, las dos máquinas indicadas deberían coincidir.

Algunos Trucos

En la Figura 4, el comando *netstat* nos muestra que, efectivamente, hemos establecido una conexión Telnet por SSH. El primer comando *netstat* nos indica que el proceso SSH con PID 3311 está escuchando en el puerto 23. El segundo comando muestra que existe una conexión al puerto 22 de este mismo identificador de proceso (3311).

Si nos fijamos con más detalle en la sintaxis usada para la apertura del túnel, nos daremos cuenta de que no es necesario que los puertos local y remoto deban coincidir. Suponiendo que la máquina remota está ejecutando un proxy configurado como proxy transparente en el puerto 3128, se podrían redirigir todas las peticiones HTTP de la siguiente manera:

```
ssh -o GatewayPorts=yes ➥
 -L 80:OrdenadorRemoto:3128 ➥
 OrdenadorRemoto
```

Este proceso de redireccionamiento de un puerto a otro es conocido como *forwarding* (reenvío) de puertos. Para permitir que otras máquinas de la red puedan usar este túnel es necesario el uso del parámetro *-o GatewayPorts = yes*.

De manera similar, los túneles pueden funcionar en dirección inversa. El siguiente comando permite el establecimiento de un túnel de retorno desde la máquina remota a nuestra máquina local:

```
ssh -R PuertoLocal:➥
OrdenadorLocal:PuertoLocal ➥
OrdenadorRemoto
```

En el ejemplo del proxy, el comando debería ser:

```
ssh -o GatewayPorts=yes ➥
-R 3128:OrdenadorLocal:80 ➥
OrdenadorRemoto
```

Túneles Gráficos

El sistema X Window ofrece de forma nativa capacidades de red, pero casi nadie hace uso de esto debido a que, otra vez, la conexión no está cifrada. La tunelización por SSH puede hacer de esta opción algo mucho más interesante.

Para realizar un túnel X11, el servicio SSH (*sshd*) debe emular un servidor X que ocupa un *display* (por defecto el 11). Cuando nos autenticamos en el servidor, *sshd* establece la variable de entorno *DISPLAY* a *localhost:11.0*. El objetivo de esto es evitar colisiones con el servidor X ejecutado de forma local. Toda información enviada a este *display* será cifrada y remitida a nuestra máquina.

OpenSUSE 11.3 tiene activado el reenvío X11 por defecto. Si fuera necesario, esta opción puede deshabilitarse desde la máquina remota, estableciendo la variable *X11Forwarding* a *no*, en */etc/ssh/sshd_config*. La variable *X11DisplayOffset* tiene un valor de 10 por defecto, y define el incremento para la variable *DISPLAY* entre el *display* virtual y el físico; no debería cambiarse este valor.

Si la máquina sobre la que vamos a realizar túneles X11 es una OpenSUSE 11.3, el fichero */etc/ssh/ssh_config* ya vendrá con la variable *ForwardX11Trusted* establecida a *yes*, lo que completaría la configuración.

Para hacer una prueba, podremos acceder a la máquina remota por medio de SSH y podríamos ejecutar el programa *Xclock*. La Figura 5 muestra el *display* (*localhost:11.0*), el proceso y las conexiones de red.

Conclusiones

El paquete SSH incluye una colección importante de aplicaciones que permiten trabajar sobre la red de forma más segura. Nos da opciones para usar desde una conexión simple cifrada, hasta el establecimiento de túneles y reenvío de puertos, pasando por conexiones X11 seguras, dejando poco que desear para el uso diario. ■

Versiones del Protocolo SSH

Las versiones actuales del protocolo SSH son 1.3, 1.5 y 2. Comparada con la versión 2, las capacidades ofertadas por la versión 1 pueden ser algo limitadas – en especial en lo referente a las opciones para la selección de algoritmos de cifrado. La versión 1 ofrece el inseguro DES o el seguro, pero lento, Triple DES (3DES). El algoritmo Blowfish es una opción rápida y (relativamente) segura. La versión 2 incluye además, entre otros, el algoritmo AES.

Otro problema que afecta a la versión 1 son las vulnerabilidades en el protocolo, que lo hacen teóricamente *hackeable*. Comparada con la versión 1, la versión 2 es algo más lenta, cosa que se nota en cuanto deseamos transferir grandes volúmenes de datos. La versión 1 se apoya en la generación de claves desde el lado del cliente durante la etapa de negociación de claves. El servidor envía su clave pública al cliente, que generará un número de 256 bits de forma aleatoria, y lo entregará de vuelta al servidor, no sin antes cifrarlo usando la clave pública del servidor.

El canal de datos es entonces cifrado usando los números aleatorios así generados. Si un programa de monitorización está escuchando la comunicación, podría obtener la clave (cifrada, eso sí). El atacante podría usar la fuerza bruta para sacarla, aunque en teoría esto es algo que podría durar varios años.

La versión 2 del protocolo se basa en el sistema de intercambio de claves de Diffie-Hellman. Éste nunca llega a transmitir la clave por el canal de comunicación. El cliente y el servidor intercambian datos de tal manera que permiten a ambos generar la misma clave de forma independiente el uno del otro.

No sería de ayuda para el atacante registrar esta información, ya que no contiene datos de utilidad para la generación de la clave. Este método de generación de claves es mucho más seguro, ya que ninguna de las partes define la clave en su totalidad.

Otra de las mejoras de la versión 2 es la capacidad del software de comprobar la integridad de los datos usando resúmenes (*hashes*) de cifrado (método MAC), en vez del inseguro CRC (Comprobación de Redundancia Cíclica). Se mejora además la capacidad de multiplexado, haciendo parecer que ambos extremos son capaces de transferir información de forma simultánea.

Todos los ejemplos de este artículo usan SSH 2, aunque algunos podrían funcionar con la versión 1.

FTP y NcFTP

REMUEVE ESOS DATOS

FTP y NcFTP son ambos clientes FTP para la línea de comandos. Mientras FTP sólo dispone de funcionalidad para la transferencia de ficheros básica, NcFTP posee un completo grupo de características adicionales. **POR HEIKE JURZIK**

FTP (File Transfer Protocol) es una serie de protocolos de Internet que ayuda a los usuarios a transferir datos entre dos ordenadores. Uno de ellos sirve de cliente FTP, mientras que la otra máquina es el servidor.

La mayoría de los servidores FTP ofrecen servicios anónimos en los que pueden introducirse *anonymous* o *ftp* como nombre de registro y no suministrar una contraseña, aunque se considera un gesto amigable introducir la propia dirección de correo. Otros servidores, ejecutados mediante suministradores de espacios web, por ejemplo, exigen introducir el nombre de usuario y la contraseña para el propósito de identificación.

Es preciso señalar que, en contraste con los protocolos de seguros (tales como Secure Shell SSH), un protocolo del tipo FTP transmite nombres de usuario, contraseñas y otros datos a través de la red en texto claro, por lo que FTP no resulta adecuado para el intercambio de información confidencial.

Si se prefiere la simplicidad de la línea de comandos y no usar el propio navegador web o el cliente FTP interno del administrador de ficheros, entonces es buena idea echarle un vistazo a las dos alternativas a la línea de comandos, FTP y NcFTP.

CONSEJO

Para cambiar de directorio en tu máquina local, usa el comando *lcd*. Para otros comandos locales, comienza el comando con un signo de exclamación. Por ejemplo, *!ls* presenta el listado del directorio local.

FTP: Toma de Contacto

FTP es un simple cliente de la línea de comandos que cualquier distribución Linux debería incluir predeterminadamente. Para abrir una conexión se hace lo siguiente:

```
ftp ftp.servidor.com
```

Alternativamente, puede arrancarse primero el cliente introduciendo *ftp*, y luego, para conectarse, se introduce el siguiente comando en la línea de comandos *ftp*:

```
ftp> open ftp.servidor.com
```

Tras establecer conexión, deberíamos apuntarnos para registrarnos. Como se ha mencionado anteriormente, puede escribirse *anonymous*, *ftp* o un nombre de usuario suministrado por el proveedor del espacio web, junto con la contraseña a juego (Listado 1). La introducción de la contraseña puede ser bastante dificultosa, ya que a diferencia de NcFTP, FTP no presenta un asterisco para cada carácter que se introduce.

Listado 1: Conectando con FTP

```
ftp> open ftp.debian.org
Connected to ftp.debian.org.
220 saens.debian.org FTP server
 (vsftpd)
Name (ftp.debian.org:huhn): ftp
331 Please specify the password.
Password:
```

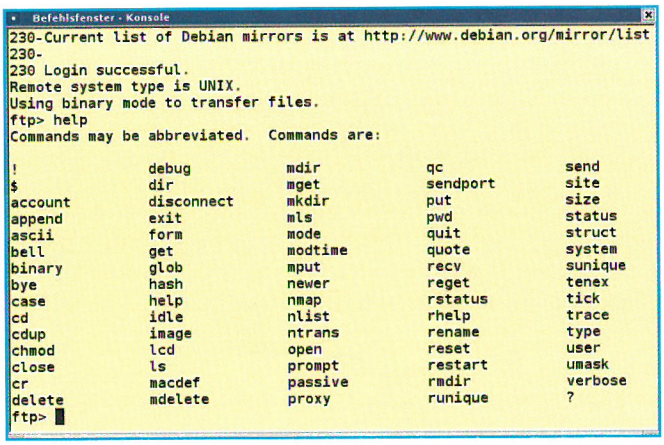


Figura 1: Listado de comandos: acceso a la ayuda online de FTP con los comandos ? o help.

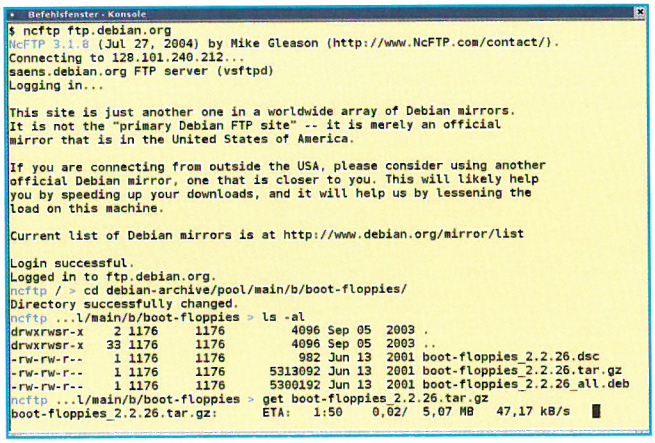


Figura 2: NcFTP muestra la ETA ("Estimated Time of Arrival" o Tiempo estimado de llegada) para un fichero concreto.

Prompt FTP

Una vez registrados, FTP presenta el prompt *ftp*, pudiéndose comenzar a introducir comandos. La salida de los comandos *help* o *?* muestra los comandos disponibles (Figura 1).

Muchos de los comandos presentan una sintaxis similar a los comandos Bash (por ejemplo, *cd*, *ls*, *exit*, etc). Algunos de éstos son auto-explicativos, mientras que otros son bastante enigmáticos. Puede pedirse al programa que muestre más información sobre cada comando mediante:

```
ftp> help <comando>
```

donde *<comando>* es el nombre de la instrucción sobre la que se recaba información.

Para descubrir en el lado del servidor el directorio actual se introduce *pwd* (print working directory), como en Bash, y con el familiar comando *ls* se presentará un listado de directorios y ficheros. Con *cd directory* podrá cambiarse a otro directorio, *cd ..* sube un nivel en el árbol de directorios y *cd* lleva hasta el directorio raíz en el servidor FTP.

get y *mget* ayudan a descargar datos del servidor FTP al directorio actual en la máquina local. Mientras que *get* espera que se introduzca el nombre de fichero completo, *mget* también soporta comodines. Para mover todos los ficheros a un directorio específico en la máquina local, se escribe *mget ** en el prompt.

Evidentemente, si se desea, se puede ser más preciso:

```
ftp> mget *.tar.gz
```

Así se descargarán ficheros con la extensión *.tar.gz*. Por defecto, FTP avisará para confirmar la descarga de cada fichero único que encaje con el modelo. Se introduce *y* o *n* para empezar la transferencia o bien para saltar un fichero, respectivamente.

El comando *prompt* ayudará a evitar cuestiones de este tipo:

```
ftp> prompt
Interactive mode off.
```

Esto también es aplicable al comando *mput*, el cual sirve para subir datos al servidor, y *mdelete*, que elimina datos del lado del servidor; ambos suponen que se dispone de acceso privilegiado de escritura (write) al servidor FTP.

El comando *close* cierra la conexión y *quit* cierra la sesión y abandona el programa.

NcFTP: ¿Hay Alguien Ahí?

Aunque la mayoría de las distribuciones no incluyen el cliente de la línea de comandos NcFTP por defecto, este cliente se encuentra habitualmente en los medios de instalación, o bien puede encontrarse descargando la herramienta desde la página de inicio de NcFTP [1], ls cual dispone de distintas versiones para varias plataformas.

Las operaciones básicas no difieren mucho de FTP. Para abrir una conexión puede especificarse un destino sobre el que se arranca el programa, o bien introducir el comando *open* en el prompt (Listado 2).

NcFTP maneja automáticamente el registro en servidores anónimos. Si se necesitan introducir nombres de usuario o contraseñas, se proporcionan los detalles cuando se ejecuta el comando:

```
ncftp> open -u nombreusuario
-p contraseña ftp.servidor.com
```

Los comandos *help*, *pwd*, *cd*, *lcd*, *close* y *quit* funcionan igual que en FTP y, evidentemente, para listar directorios se usará *ls*. Sin embargo, NcFTP presenta aquí una característica adicional; suponiendo que el servidor soporta esto, *ls -l* ofrecerá un listado extendido, similar a *ls -la*, que presentará ficheros ocultos y directorios.

NcFTP tiene más que ofrecer, ya que el cliente soporta completado con tabulador, con el que se estará familiarizado desde Bash. Pueden introducirse un par de letras de un comando o el nombre de un fichero/directorio y pulsar [Tab]. NcFTP completará el nombre si es único. Escribiendo las dos primeras letras de un comando y pulsando la tecla tabulador se le dice que presente todos los comandos que comienzan con esa letra.

NcFTP: Nuevos Amigos

Como probablemente era de suponer, el comando para la descarga de ficheros a la máquina local es *get*. NcFTP añade una característica muy conveniente que muestra el progreso de descarga y el rango de transferencia en la parte inferior de la pantalla (Figura 2).

Otra práctica característica es la habilidad de reanudar descargas canceladas. Cuando se

Listado 2: Abriendo una Conexión

```
huhn@asteroid:~$ ncftp
Ncftp 3.1.8 (Jul 27. 2004) by
Mike Gleason
 (http://www.Ncftp.com/contact/).
ncftp> open ftp.debian.org
Connecting to 128.101.240.212...
saens.debian.org FTP server
(vsftpd)
Logging in...
```

ejecuta *get* contra un fichero que ya existe en la máquina local, NcFTP nos dice que *The local file filename already exists*, es decir, que el nombre del fichero local ya existe.

El cliente FTP presenta entonces el tamaño del fichero y la marca horaria, detalles para el fichero local y el remoto (suponiendo que ambos son diferentes).

Seguidamente, el programa pregunta sobre qué hacer a continuación. Como puede apreciarse en la Figura 3, los usuarios disponen en este punto de determinadas opciones:

- *o* – Sobreescribir el fichero local.
- *r* – Reanudar una descarga cancelada.
- *a* – Añadir el fichero desde el servidor FTP al fichero local.
- *s* – Cancelar la conexión.
- *n* – Guardar el fichero bajo un nuevo nombre.
- *o!* – Sobreescribir todos los homólogos locales en descarga.
- *r!* – Continuar la transferencia de ficheros para todos los ficheros.
- *s!* ó *c* – Cancelar la acción para todos los procesos.

NcFTP: Flexible

Al igual que *ls*, *get* posee determinadas opciones con características adicionales. Para transferir un fichero completo recursivamente (es decir, incluyendo todos sus ficheros y subdirectorios), tendremos que hacer lo siguiente:

```
ncftp> get -R directorio
```

Además, *get* posee un parámetro que permite a los usuarios almacenar un fichero con un nombre diferente. El comando siguiente:

```
ncftp> get -z README README.TXT
```

almacenará el fichero README como README.TXT localmente.

Adicionalmente, *get* dispone de otra práctica característica, especialmente para ficheros de texto: por defecto, NcFTP usará modo de transferencia binario (lo que significa que cambiará el contenido del fichero cuando se transfiera). Si el fichero es un fichero ASCII con diferentes finales de caracteres de línea para editores en diferentes plataformas, el comando siguiente:

```
ncftp> get -a
file.txt
```

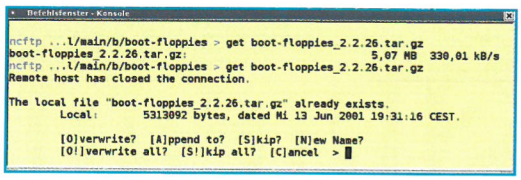


Figura 3: Si intentas descargar un fichero que ya existe localmente, NcFTP te preguntará por el paso siguiente.

le dirá a NcFTP que descargue el fichero en modo ASCII. Para cambiar a modo ASCII permanentemente se ejecuta el comando *type*:

```
ncftp> type ascii
ncftp> type
Type is A (ASCII).
ncftp> type binary
ncftp> type
Type is I (binary/image).
```

Como puede apreciarse, *type* no sólo cambia de modo, sino que muestra el modo actual.

Por último, el comando *get* tiene un parámetro *-f* que le dice al cliente que sobreescriba los ficheros existentes, sin pedirle al usuario que reinicie la descarga, ni nada por el estilo.

NcFTP: Background

Algunos comandos NcFTP ejecutan automáticamente acciones cliente como procesos de fondo. El uso de los comandos *bgget*, *bgput* o *bgstart* ofrecen la ventaja de dejar liberar inmediatamente el prompt del cliente FTP para más comandos.

Primero debe decírsele al programa la dirección de transferencia y los datos fuente mediante la ejecución de los comandos *bgget* y *bgput* (Listado 3). De nuevo, ejecutando el comando *(bg)put* se suponen privilegios de escritura (write) en el servidor.

El comando *jobs* presenta los trabajos de transferencia que se encuentran en espera en cola. La transferencia puede comenzarse introduciendo el comando *bgstart*. NcFTP empezará entonces a procesar los trabajos en el background (incluso después de que se abandone el programa), y almacenará los resultados en un fichero de registro:

```
Watch the "/home/
huhn/ .ncftp/spool/
log" file to see
how it is progressing
```

tail -f es útil para monitorizar el fichero de registro, porque actualizará la presentación siempre que el fichero cambie.

Esto permite monitorizar el progreso de NcFTP sin necesidad de ejecutar los paginadores *less* o *more* múltiples veces.

NcFTP: Marcas

Para evitar escribir, o para refrescar la memoria, pueden usarse marcas en NcFTP. Para ello se escribe *bookmark* durante una sesión y se pulsa [Enter] para confirmar el nombre sugerido por NcFTP, o se introduce el nombre elegido para la marca:

```
ncftp> bookmark
Enter a name fo this
bookmark, or hit enter
for "debian": debiantest
Bookmark "debiantest" saved.
```

El clientre NcFTP también permite guardar una marca cuando se introducen *close* o *quit*. De hecho, NcFTP incluso soportará cuentas de protección de contraseñas, almacenando (encriptadas) los credenciales bajo el directorio de inicio del usuario en *.ncftp/bookmarks*.

La próxima vez que se abra la misma conexión se introduce *open* y se le pasa el nombre de la marca. Alternativamente, se introduce *open* y se pulsa [Enter] para presentar una lista de marcas. A propósito, el comando *bookmarks* también presentará una salida similar.

En este modo, el cliente FTP incluso ofrece un editor simple que puede ser usado para editar las marcas propias:

- [Enter] – Abre una conexión a la dirección especificada.
- */ed* – Edita la entrada especificada.
- */del* – Elimina la marca.
- */dup* – Copia la marca.
- */new* – Añade una marca.

Una vez que se han editado las marcas, puede pulsarse [X] para abandonar el editor de marcas. ■

Listado 3: bgget

```
01 ncftp ...e/pool/main/m/man2html > bgget *.deb
02 + Spooled: get man2html_1.5.-23.1_alpha.deb
03 + Spooled: get man2html_1.5.-23.1_arm.deb
04 + Spooled: get man2html_1.5.-23.1_i386.deb
05 + Spooled: get man2html_1.5.-23.1_m68k.deb
06 + Spooled: get man2html_1.5.-23.1_powerpc.deb
07 + Spooled: get man2html_1.5.-23.1_sparc.deb
```

RECURSOS

[1] Sitio web de NcFTP: *http://www.ncftp.com*

Sincronización de Ficheros con Rsync

SINCRONÍZATE

Rsync permite sincronizar datos tanto en un ordenador local como remoto. Incluso puede ser usado con SSH para el cifrado de información. **POR HEIKE JURZIK**

Rsync permite mantener los datos actualizados y ayuda a controlar los confusos cambios que se producen en las nuevas versiones de cada fichero. La herramienta de sincronización Rsync ofrece opciones que ayudan a administrar las propiedades de ficheros, funciona bien con SSH y es perfecta para la transferencia rápida de grandes volúmenes de datos si el ordenador objetivo ya posee una copia de una versión previa. Rsync comprueba las diferencias entre la fuente y las versiones objetivo de los datos y transfiere solamente aquellos que hayan cambiado.

Sincronicity

Rsync compara los datos en dos máquinas. La sintaxis genérica para esto es *rsync [opciones] fuente objetivo*. La elección de la fuente y el objetivo es crítica. Es preciso decidir cuidadosamente la dirección donde se va a realizar la sincronización para evitar la pérdida de datos. El cuadro "Datos Unison" presenta una alternativa a esta salida de un solo sentido basada en la herramienta *Unison*, la cual permite sincronizar en ambas direcciones.

Rsync ofrece un apoyo importante para la monitorización de la transferencia de datos: si se invoca a Rsync con la etiqueta *-n*, el programa presentará una ejecución de prueba:

```
$ rsync -n dir1/* dir2/
skipping directory dir1/dir1
skipping directory dir1/dir2
skipping non-regular file➥
"ficherol.tex"
kap01.tex
kap02.tex
kap03.tex
...
```

Tal y como muestra la salida, Rsync transferirá ficheros y directorios pero omitirá los enlaces simbólicos. La herramienta hace referencia a estos últimos como ficheros *no-estándar* (*non-regular*). Para transferir subdirectorios recursivamente hasta el nivel

Datos Unison

Unison [1] es otra importante herramienta de sincronización, que no sólo sincroniza en ambas direcciones, sino que también soporta sistemas operativos no Linux. Este programa se encuentra disponible para Linux, Windows, Mac OS X y Irix, tal y como puede comprobarse si se echa un vistazo al área de descarga de su sitio web.

Unison puede ejecutarse desde la línea de comandos o desde su GUI. Para sincronizar dos directorios en la máquina local se escribe: unison -ui text carpeta1 carpeta2, por ejemplo.

Si difiere el contenido de las dos carpetas, Unison dará a conocer la dirección de sincronización y decirnos si el fichero es nuevo o ha sido modificado.

```
carpeta1    carpeta2
new file -->    kap01.
tex  [f]
<-- changed     kap02.
tex  [f]
```

Para aceptar esta sugerencia se pulsa la tecla [F]. Si se presenta un conflicto que Unison no puede resolver porque, digamos, las dos versiones del fichero han cambiado desde la última vez que se ejecutó el programa, Unison presenta un signo de interrogación <-?-> invitando a tomar una decisión. Se puede escribir < ó > o pulsar [Shift-7] (/) para posponer la decisión hasta otro momento, o pulsar [Shift-I] para posponerla para siempre.

Para ver los comandos disponibles puede escribirse un signo de interrogación (?). Al final, el programa comprueba una vez más si realmente se desean aplicar los cambios: pulsando [Y] se arranca la operación de sincronización.

```
Proceed with➥
propagating updates? [] y
Propagating updates
UNISON started propagating ➥
changes at 20:20:34 ➥
on 28 Feb 2006
[BGN] Copying ficherol.tex
  from /home/huhnix/dir1
  to /home/huhnix/dir2
...
Saving synchronizer state
Synchronization complete
```

Igual de simple es la sincronización con máquinas remotas. Primero hay que asegurarse de que la versión Unison instalada es la misma en ambas máquinas y luego se escribe unison -ui text carpeta1 ssh://usuario@maquinaobjetivo/carpeta.

Como puede observarse, la sintaxis es un poco distinta a la de Rsync. En lugar de los dos puntos se tiene una barra. Para usar una ruta absoluta es preciso añadir otra barra: *unison -ui text carpeta1 ssh://usuario@maquinaobjetivo/carpeta.*

más bajo, habrá que especificar la opción *-r*. Adicionalmente, si se añade la etiqueta *-l* recoge los enlaces simbólicos. Evidentemente es posible una combinación de estas opciones.

```
rsync -rl dir1/* dir2/
building file list ... done
ficherol.tex ->➤
ficherol_nuevo.tex
kap01.tex
...
```

Rsync dispone de una método alternativo para manejar enlaces simbólicos: si se sustituye *-l* por *-L*, el programa resolverá el enlace, y los enlaces simbólicos acaban como ficheros normales en el directorio objetivo.

Barra Libre

Al añadir una barra al nombre del directorio se modifica el modo en el que Rsync desarrolla una operación. Por ejemplo, si se da al comando *rsync -a fuente/carpeta objetivo*, Rsync transferirá el directorio *carpeta* y su contenido al directorio objetivo.

Si se añade una barra al directorio objetivo, rsync se limitará a transferir el contenido de *carpeta* al objetivo, metiéndolos directamente en el directorio objetivo (Figura 1).

Volver a Empezar

Si se usa Rsync para crear copias de seguridad, tiene sentido mantener los atributos de los ficheros originales. Los atributos incluyen permisos de lectura, escritura y ejecución junto con marcas temporales, es decir, información de la última hora y fecha de acceso (atime), del último cambio de estado (ctime) y la última modificación (mtime).

Además, los administradores pueden aprovechar los parámetros que preservan los datos del propietario y del grupo y soportar ficheros de dispositivo. Para mantener los permisos se especifica la opción *-p*, *-t* se encarga de las marcas temporales y *-g* mantiene intacto el atributo del grupo.

Si bien cualquier usuario normal puede especificar los parámetros mencionados, las opciones *-o* (guarda los datos propietarios) y *-D* (atributos de dispositivo) sólo se encuentran disponibles para el superusuario. La línea de comandos completa con todas estas opciones sería algo como *rsync -rlptgoD home/huhn/*backup/*. Pero existe un atajo: Rsync dispone de una opción especial que combina estos parámetros para este caso. En vez de *-rlptgoD* se escribe *-a*.

Exclusiones

Rsync dispone de una práctica opción que permite excluir determinados ficheros del proceso de sincronización. Para utilizarlo, se especifica la opción *–exclude=* y una pauta de búsqueda y se definen los ficheros a excluir. Esta opción permite el uso de comodines: *rsync -a –exclude=*bak ~/article/*backup/*.

Esto excluye ficheros que acaban en *.bak*. Si se desean excluir ficheros acabados en tilde se añade otro *–exclude*: *rsync -a –exclude=*bak –exclude=*~ fuente objetivo*.

Puede ahorrarse tiempo almacenando las exclusiones en un fichero de texto. Se requiere una línea de separación para cada modelo de búsqueda. Hay que especificar el parámetro *–exclude-from=FicherodeExclusion* para analizar el fichero.

Cooperación

Rsync incluso puede sincronizar datos a través de una red. Para encriptar y transferir ficheros usando SSH se pasa el parámetro *-e ssh*. Es preciso cambiar la fuente y el objetivo con el nombre o dirección IP de las máquinas local y remota. Si el nombre de la cuenta de la máquina remota es distinta, hay que añadir el nombre de usuario remoto seguido de un signo arroba (@), de la siguiente manera: *rsync -e sshb fuente usuario@hostobjetivo:objetivo/*.

Se escribe una ruta relativa al propio directorio de inicio o bien una ruta absoluta después de los dos puntos. Una variable de entorno evita tener que introducir los parámetros para la encriptación SSH cada vez que se ejecuta el comando. Para Bash el comando es: *RSYNC_RSH=ssh*.

Debido a que la variable solamente es válida dentro del contexto de la shell actual, hay que añadir la línea de exportar el fichero de configuración de tu Bash, *~/.bashrc*, asignar la variable permanentemente y luego ejecutar *source ~/.bashrc* para reanalizar el fichero.

Amontonar Datos

Rsync es perfecto para la transferencia de grandes volúmenes de datos. Si se especifica el parámetro *–partial* y se interrumpe la transferencia, puede tomarse dicha transferencia a partir del punto en la que se dejó. Especificando la opción *–progress* se obtiene un indicador de progreso que permite seguirle la pista a la operación de transferencia.

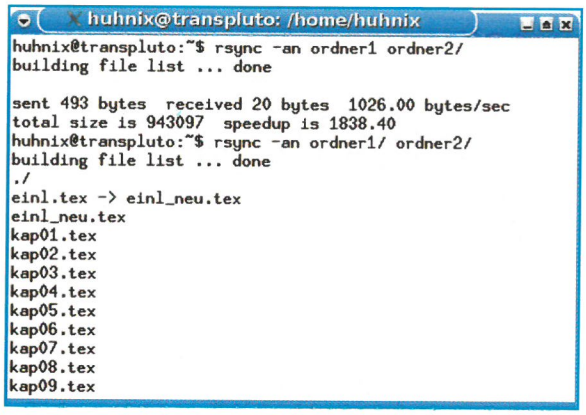


Figura 1: La barra es importante: ¿Copiará Rsync los contenidos o el fichero en sí?

```
$ rsync –progress –partial➤
debian.iso transpluto:➤
/scratch/debian/
Password:
debian.iso
    22543992   3%   5.15MB/s
0:02:14
```

Al otro lado de la conexión, el fichero parcial se mantiene oculto en el directorio objetivo. Si se escribe *ls -a*, se puede ver un fichero con un nombre como *.debian.iso.wtKNr7*. El punto al comienzo del nombre del fichero mantiene al fichero oculto y la extensión arbitraria minimiza el peligro de sobreescritura de los ficheros existentes.

Cuando se completa la transferencia, el fichero recobra su nombre original. Si se interrumpe puede ser restablecida especificando nuevamente la opción *--partial*.

La opción *--partial* tiene una desventaja. Si se usa Rsync para actualizar un fichero existente y se interrumpe la transferencia, desaparece el fichero original de la máquina objetivo y es sustituido por el fichero parcial que se ha transferido. Este problema puede solucionarse creando un enlace duro al fichero original primero: *ln debian.iso debian_orig.iso*. Si la transferencia falla, no se perderá la imagen ISO. En vez de ello, aparecerá un nuevo nombre al fichero parcial, pero sin destruir el original. ■

RECURSOS

[1] Página de inicio de Unison: *http://www.cis.upenn.edu/~bcpierce/unison/*

El brillo del sol, una piscina, un portátil, una WLAN abierta y las herramientas inalámbricas de la línea de comandos es todo cuanto necesitas para enviar datos desde cualquier lugar. **POR HEIKE JURZIK**

Wireless Tools [1] es una completa y poderosa colección de programas de la línea de comandos que no solamente sirve información sobre WLANs, sino que también ayuda a configurar conexiones inalámbricas.

Las Herramientas

El juego de herramientas incluye lo siguiente:

- *iwconfig*: Presenta información sobre configuración y configura interfaces WLAN
- *iwlist*: Presenta información sobre nuestras opciones y WLANs en los alrededores
- *iwgetid*: Presenta información de las interfaces existentes, igual que *iwconfig*, aunque no ofrece opciones de configuración
- *iwpriv*: Configura varios parámetros específicos de driver
- *iwspy*: Recopila información de otros clientes WLAN
- *iwevent*: Controla interfaces WLAN

La Suite

Probablemente encontraremos esta colección de programas en los repositorios de nuestra distribución en el paquete de herramientas inalámbricas. Los ordenadores con una interfaz WLAN funcionando ya tendrán el paquete preinstalado.

En algunas distribuciones, la suite Wireless Tools incluye adicionalmente el programa *ifrename*. Los usuarios de Debian pueden instalar la herramienta desde un paquete del mismo nombre. Para usar los programas preinstalados deberemos instalar y cargar los drivers para nuestra tarjeta WLAN.

Listado 1: dhclient

```
01 # dhclient eth0
02 ...
03 DHCPDISCOVER on eth0 to
   255.255.255.255 port 67
   interval 8
04 DHCPDISCOVER on eth0 to
   255.255.255.255 port 67
   interval 19
05 DHCPOFFER from 192.168.1.254
06 DHCPREQUEST on eth0 to
   255.255.255.255 port 67
07 DHCPACK from 192.168.1.254
08 bound to 192.168.1.72 -
   renewal in 38488 seconds.
```

iwconfig

iwconfig funciona como **ifconfig**. Si se le llama sin ningún otro parámetro nos dará una conjunto completo de información acerca de nuestras interfaces de red en una ventana de terminal.

La Figura 1 muestra la salida en un portátil, con *lo* como la interfaz bucle, *eth1* como la tarjeta de red normal y *eth0* como una interfaz WLAN.

Se ha establecido una conexión a una red llamada *cretablue*, el nombre de la tarjeta WLAN es *Broadcom 4318*. El punto de acceso para la red cretablue nos dice su dirección MAC y que está ejecutándose en modo administrado.

Si preferimos no ver la información para las otras interfaces, pasamos el nombre de la interfaz en la que estamos interesados a *iwconfig*, por ejemplo:

```
/sbin/iwconfig eth0
```

El administrador del sistema *root* puede ejecutar el comando sin especificar la ruta porque éste reside en */sbin*, que es la ruta de root.

Configuración de una Interfaz

Además del nombre de la interfaz de red, *iwconfig* soporta otras opciones de la línea de comandos, las cuales siguen al comando mostrado previamente. Los comandos siguientes cambian una configuración ya existente, requiriendo privilegios root. Para asociarnos con otra red deberemos especificar el **ESSID**, que sigue al comando iwconfig y al nombre de la interfaz:

```
iwconfig eth0 essid ➥
"Beach Cafe"
```

GLOSARIO

ifconfig: Una de las herramientas de configuración de red más importantes. Cuando se llama sin ningún parámetro adicional, *ifconfig* mostrará las interfaces de redes configuradas actualmente. La mayoría de las distribuciones almacenan el programa en el directorio */sbin*, de modo que los usuarios normales deberán especificar la ruta completa, */sbin/ifconfig*. Los administradores pueden usar *ifconfig* para configurar las tarjetas de red desde la shell.

Adonde vayas...

Lee Linux Magazine desde cualquier sitio con nuestra subscripción digital.
Accede a todos los números en PDF entrando en nuestro sitio.
Encuentra lo que buscas con un sencillo formulario de búsqueda.
Mantén tu propia biblioteca de artículos y léelos desde cualquier dispositivo.

... Linux Magazine va contigo.

http://www.linux-magazine.es/digital

Si el nombre de la red incluye blancos o caracteres especiales, necesitaremos colocarlos entre dobles comillas. Si la red usa una clave WEP, lo especificaremos con la opción *key*. Para ello introducimos la llave en notación hexadecimal o escribimos *s:* seguido de la contraseña WEP:

```
iwconfig eth0 essid ➥
"Beach Cafe" ➥
key s:IceCreamRocks
```

Si fuera necesario, también podemos definir el canal que queremos usar (*channel11*), especificar el modo (por ejemplo, *mode Managed* o *mode Ad-Hoc*, etc), especificar el nick que usaremos en la red (por ejemplo, "External Cisco Card"), y muchas otras cosas. La página man de *iwconfig* tiene una lista muy completa. No debemos olvidar configurar la dirección IP para nuestra tarjeta. La mayoría de las redes públicas soportarán asignaciones de direcciones IP a través de DHCP; root puede usar el programa *dhclient* para esto. Si fuera necesario, podemos pasarle un nombre de interfaz a esta herramienta (Listado 1).

Rastreador

iwlist es otra útil herramienta. Podemos usar la opción *scanning* con ella para buscar redes inalámbricas disponibles (Listado 2).

Si estamos más interesados en las frecuencias y canales que soporta nuestra tarjeta WLAN, introducimos la opción *frequency*. Otros parámetros incluyen la tasa de bits soportados (*iwlist bit rate*) o la potencia del transmisor (*iwlist power*).

iwgetid es un tipo de *iwconfig* de peso ligero. Por defecto, da una única línea que nos dice qué interfaz está asociada con (E)SSID:

```
# iwgetid
ath0     ESSID:"cretablue"
```

iwgetid es por ello perfecto para scripting.

Otras opciones distintas le dirán al programa que presente otros detalles. Por ejemplo, *--ap* nos da el punto de acceso a la dirección MAC, *--freq* muestra la frecuencia, *--channel* nos da el canal, etc.

En Escucha

iwevent ofrece un tipo de logfile para eventos inalámbricos y controla la actividad en la red inalámbrica por defecto (véase el Listado 3).

Para detener la escucha pulsamos Ctrl + C. Si necesitamos información sobre los usuarios de otra WLAN en el mismo radio podemos usar *iwspy*. Para hacerlo debemos hacer ping a las direcciones IP,

```
$ ping 192.168.1.100
PING 192.168.1.100➥
(192.168.1.100) 56(84) bytes➥
of data.
64 bytes from 192.168.1.100:➥
icmp_seq=1 ttl=64 time=6.23 ms
...
```

a continuación pasamos nuestro propio interfaz y la dirección IP de nuestro ordenador remoto:

```
iwspy ath0 192.168.1.100
```

iwspy nos dará entonces los detalles sobre la calidad de la conexión, la fuerza de la señal, la dirección MAC de otro cliente, etc (Listado 4).

Todo Incluido

Como podemos ver, no es necesitario disponer de complicadas y vistosas herramientas GUI para establecer nuestra configuración de red y mantener una conexión inalámbrica desde Grecia con la oficina. La suite Wireless Tools proporciona a los seguidores de la línea de comandos todo cuanto pudieran necesitar durante sus bien merecidas vacaciones.

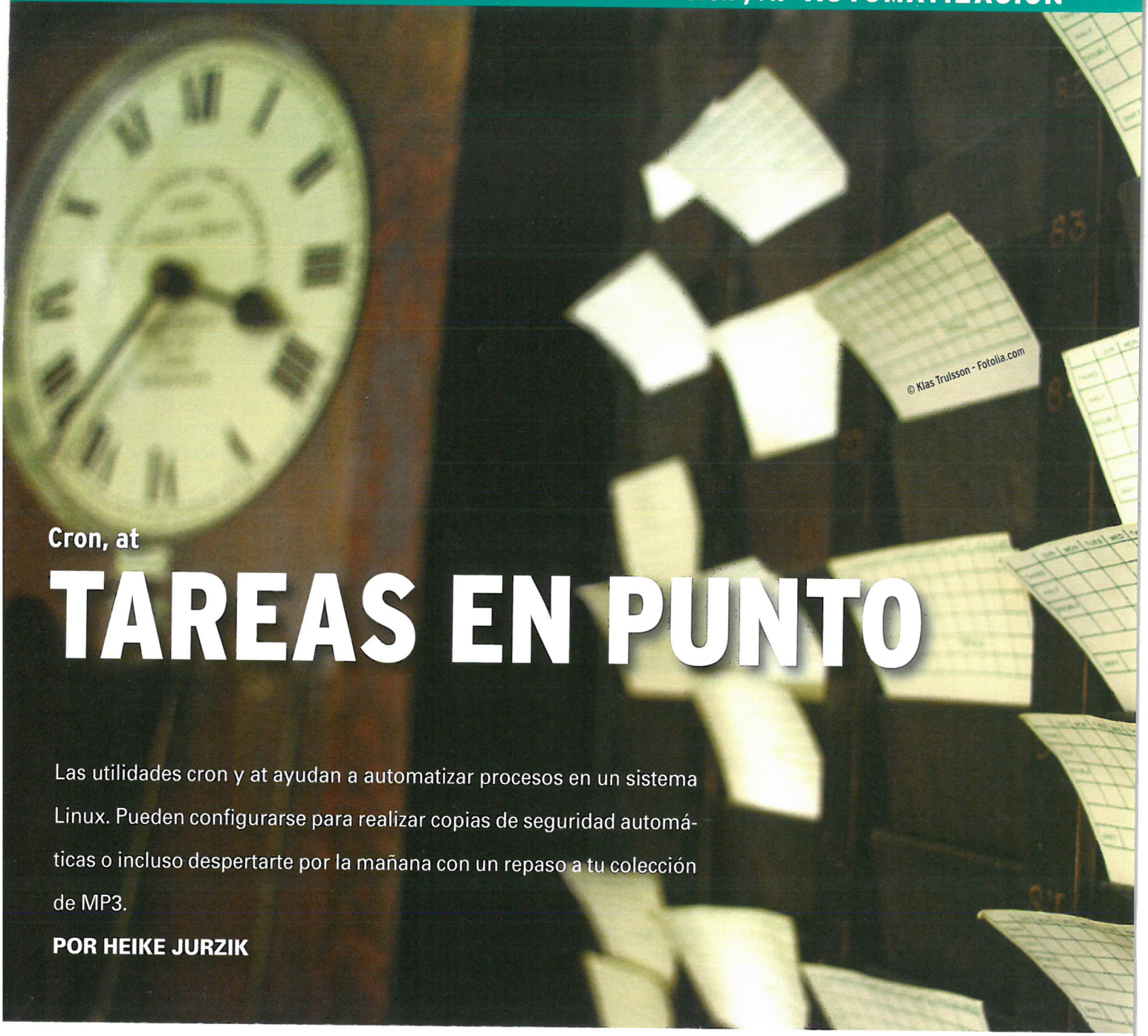
Después de encontrar la mejor red disponible en los alrededores y establecer una conexión, los programas nos permiten incluso no perder de vista a nuestros compañeros veraneantes. ■

Listado 2: Opción de Escaneado iwlist

```
01 # iwlist scanning
02 eth0 Scan completed :
03         Cell 01 - Address: 00:14:6C:91:77:EC
04                   ESSID:"Poolbar upstairs"
05                   Protocol:IEEE 802.11bg
06                   Mode:Master
07                   Channel:6
08                   Encryption key:off
09                   Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 9 Mb/s; 11 Mb/s
10                             6 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
11                             48 Mb/s; 54 Mb/s
12                   Quality=100/100   Signal level=-217 dBm
13                   Extra: Last beacon: 884ms ago
14         Cell 02 - Address: 02:20:A6:B7:04:41
15                   ESSID:"STAR_BEACH"
16                   Protocol:IEEE 802.11bg
17                   Mode:Master
18                      Channel:11
19                      Encryption key:off
20                   Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 22 Mb/s
21                   Quality=100/100   Signal level=-145 dBm
22                   Extra: Last beacon: 232ms ago
```

GLOSARIO

ESSID: Extended Service Set Identifier. Un identificador único para una red inalámbrica IEEE 802.11 con un máximo de 32 dígitos. El ESSID elimina la ambigüedad de múltiples redes inalámbricas en el mismo lugar.



Cron, at

TAREAS EN PUNTO

Las utilidades cron y at ayudan a automatizar procesos en un sistema Linux. Pueden configurarse para realizar copias de seguridad automáticas o incluso despertarte por la mañana con un repaso a tu colección de MP3.

POR HEIKE JURZIK

En Linux no hay necesidad de tomar notas para recordar tareas que necesitas llevar a cabo en el futuro. El programa *at* permite programar las tareas a realizar, mientras que *cron* controla los trabajos recurrentes.

Un demonio corre en el fondo para asegurarse de que las tareas se realizan según la programación establecida. El demonio comprueba los trabajos nuevos una vez por minuto. El que corresponde a at se denomina *atd*, mientras que el de cron es conocido como *cron(d)*.

At Tu Servicio

Puede invocarse a *at* desde la línea de comandos y esperar a que lleve a cabo su trabajo. A continuación se escriben más comandos en la shell y se sale presionando [Ctrl] + [D]:

```
$ at 07:00
warning: commands will ➚
be executed using /bin/sh
at> mpg321 -z /home/huhn/mp3/*
at> <EOT>
job 6 at 2006-01-03 07:00
```

De esta manera se le está diciendo al reproductor de la línea de comandos Mpg321 que nos despierte a las 7 en punto reproduciendo una selección de canciones del directorio */home/huhn/mp3/*. Para que esto funcione, lógicamente, el ordenador debe estar encendido a la hora designada.

La Tabla 1 ofrece los formatos más comunes para la hora. Se observa que *at* es persistente, es decir, que continuará corriendo después de reiniciar la máquina y recordará la programación durante los próximos seis meses.

Después de completar la tarea, esta utilidad envía un mensaje de correo a su propietario con el estado del trabajo, no importa si dicho trabajo se llevó a cabo con éxito o no. Para ello es necesario que la configuración obtenida del correo funcione correctamente (al menos para los envíos locales). Para comandos que no

Listado 1: Entradas Crontab de Debian

```
17 *  * * *  root   run-parts – report /etc/cron.hourly
25 6  * * *  root   test -x /usr/sbin/anacron || run-parts – report /etc/cron.daily
47 6  * * *  root   test -x /usr/sbin/anacron || run-parts – report /etc/cron.weekly
52 6  * * *  root   test -x /usr/sbin/anacron || run-parts – report /etc/cron-monthly
```

producen salidas por defecto, tales como *rm*, *mv* o *cp*, puede forzarse un mensaje de correo. Para hacerlo, se configura el parámetro *-m* en *at* de la siguiente manera:

```
at -m 13:31
```

Mostrando y Borrando Tareas

Los comandos at programados se almacenan en la cola. Ésta puede verse en la pantalla si se invoca a *at -l* o *atq*:

```
$ atq
7     2006-01-03 07:00 a huhnix
11    2006-01-03 12:00 a huhnix
12    2006-06-09 14:24 a huhnix
```

Desafortunadamente, *at* no es muy comunicativo en este sentido, pero ofrece el número de tarea al comienzo de la línea, la fecha y la hora, el nombre de la cola (*a*) y el nombre del usuario. No dice nada de los trabajos que se encuentran programados. Además, sólo permite ver los trabajos propios como usuario normal. Como se habrá podido adivinar, el único que puede obtener una lista completa de los trabajos programados es el administrador del sistema.

Si se desean más detalles sobre lo que nos depara el futuro hay que cambiar a superusuario y modificar todo el directorio de trabajo de *at* bajo */var/spool/atjobs/* (para Suse Linux), o */var/spool/cron/atjobs/* (para Debian). Los ficheros de texto dicen exactamente qué comandos correrán .

Para borrar una tarea, se escribe el comando *at -d* o *atrm*, especificándose el número del trabajo:

```
$ atrm 7 11
$ atq
12    2006-06-09 14:24 a huhnix
```

Privilegios de Acceso

Existen dos ficheros que controlan quiénes pueden trabajar con *at*. Uno de ellos es */etc/at.allow*, y el otro */etc/at.deny*. La mayoría de las distribuciones tienden a disponer de un fichero *at.deny* con unas cuantas entradas "pseudo-usuario" para *lp* (el demonio de impresión) o *mail* (para el demonio de correo). Si se crea un fichero *at.allow* como superusuario, se necesitan entradas para todos los usuarios que tienen permitido correr trabajos *at*. En este caso, *at.deny* no se analiza.

Un Cron para todas las Estaciones

Si se desean manejar tareas recurrentes de forma regular, no es recomendable correr *at* repetidamente. En su lugar, debería probarse con la otra opción que brinda Linux, *cron*. Este comando también corre en el fondo trabajos a intervalos regulares. De nuevo, el programa necesita de la máquina encendida, y reprograma su "agenda" cuando se inicializa el ordenador. De hecho, los dos programas tienen más cosas en común, y así, *cron*, al igual que *at*, envía un correo a la cuenta propietaria para confirmar que un trabajo se ha completado con éxito.

Las tareas individuales son referidas como "trabajos cron" y se administran en la etiqueta cron. Existe una tabla con seis columnas que define el momento en el que ha de llevarse a cabo un trabajo específico. Cada comando ocupa en dicha etiqueta una línea única. Los primeros cinco campos definen la hora, mientras que el sexto contiene el programa que va a ejecutarse, incluyendo sus parámetros.

Como usuario normal, se puede crear una etiqueta cron en la línea de comandos ejecutando el programa crontab. El comando correspondiente para realizarlo es *crontab -e*, donde el parámetro *-e* indica que la tabla se está editando.

Adicionalmente, como administrador del sistema pueden modificarse los crontabs de cualquier usuario si se especifica el parámetro *-u* y se proporciona el nombre de la cuenta:

```
crontab -u huhn -e
```

Por defecto, esto llama al editor vi. Si se prefiere otro editor de texto diferente, se configura la variable de entorno *$EDITOR* para que quede reflejado, tal y como sigue:

```
export EDITOR=/usr/bin/kwrite
```

Para que este cambio permanezca siempre, se añade esta línea al fichero de configuración Bash y se reanaliza introduciendo *source ~/.bashrc*.

Tabla 1: Formatos de Hora para at

Formato	Significado
16:16	16:16 horas de hoy.
07:00pm	19:00 horas de hoy (si no se especifica am o pm, se supone que se trata de am.
now	Ahora
tomorrow	Mañana
today	Hoy
now+10min	En diez minutos; también pueden especificarse horas, días, semanas y meses.
noon tomorrow	A las 12:00 del próximo día; también puede tratarse del *teatime* (=4:00 pm) o de *midnight* (medianoche).
6/9/06	9 de Junio del 2006, notaciones alternativas para la fecha son, por ejemplo, 6.9.06 y 6906.

Bien Estructurado

Las líneas de crontab no pueden contener órdenes que abarquen más de una línea. En el fichero, los comentarios se indican mediante una almohadilla (#) al comienzo de la línea. Los seis campos contienen la información siguiente, y en este orden:

- Minuto: Son posibles valores desde 0 a 59, así como el comodín *
- Hora: Valores entre 0 y 23 ó *
- Día: Valores desde 1 a 31 ó *
- Mes: De 1 a 12, de *Jan* a *Dec*, *jan* a *dec* o *
- Día de la semana: De 0 a 7 (donde tanto 0 como 7 significan Domingo), *Sun* a *Sat*, *sun* a *sat* o *
- Comando: El comando a correr incluyendo sus opciones. Alternativamente puede ser el nombre de un script con más comandos.

Si se desea que el ordenador nos despierte a las siete cada mañana, se añade la siguiente entrada al crontab:

```
0 7 * * * mpg321 -z ➥
/home/huhn/mp3/*
```

En los campos individuales los valores pueden estar separados por una coma. Para que los sábados y domingos no nos despierte la alarma, se añade al quinto campo correspondiente al día de la semana lo siguiente:

```
0 7 * * 1,2,3,4,5 mpg321 -z ➥
/home/huhn/mp3/*
```

También es posible especificar un rango usando guiones (*1-5*). Pero si se usan nombres de los días de la semana es más fácil leer las entradas:

```
0 7 * * mon-fri ...
```

También puede ser útil una combinación de veces. Si el cuarto campo (para el mes) tiene valores de *1-4*, *7*, *10-12*, debe interpretarse "de Enero a Abril", "Julio", "Octubre a Diciembre" respectivamente. Una barra seguida de un número también define períodos de tiempo regulares; por ejemplo, */2* en la segunda columna significa "cada dos horas" y *1-6/2* "1,3,5".

Las tablas de cron para los usuarios se almacenan en el directorio */var*. Cada distribución tiene una propuesta diferente para la organización de las tablas. Por ejemplo, Debian las almacena en */var/spool/cron/crontabs/* y las clasifica por el nombre de usuario; en cambio, Suse Linux usa el directorio */var/spool/cron/tabs/*.

Como no se dispone de permiso de lectura para este directorio como un usuario normal, puede presentarse una tabla de cron propia en la línea de comandos ejecutando el programa crontab:

```
$ crontab -l
10 8 * * mon-fri➥
 mpg321 -z ➥
/home/huhn/mp3/*
```

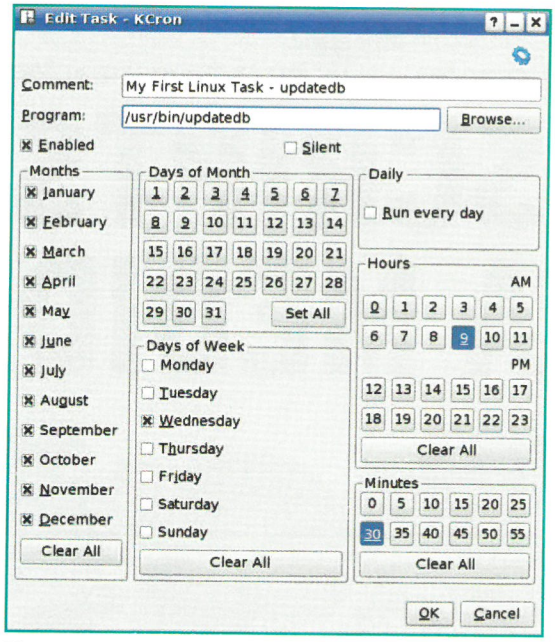


Figura 1: Kcron proporciona una GUI muy cómoda para administrar la configuración de cron.

Para borrar entradas individuales, se arranca de nuevo el editor introduciendo *crontab -e*; si se desea borrar la tabla completa, puede ejecutarse en su lugar *crontab -r*.

Tablas de Cron Globales

Cron no maneja solamente listas de usuarios específicos, sino que también ayuda a los superusuarios en las tareas de administración del sistema. Trabajando como superusuario, se aconseja echar un vistazo al fichero */etc/crontab*, éste dirá los trabajos que maneja cron. Dependiendo de la distribución, el cron global puede variar; las entradas de Debian se muestran en el Listado 1, por ejemplo.

A diferencia de lo que ocurre con usuarios crontabs normales, el crontab global tiene un séptimo campo que contiene el nombre del usuario con los privilegios que correrá el comando (esto es algo muy típico de root). El Listado que se ve más abajo nos dice que el demonio cron corre las *run-parts--report /etc/cron.hourly* con privilegios de superusuario, una vez cada hora a los 17 minutos de cada hora, mientras que a las 6:52 am del primer día de cada mes corre *runs-parts--report/etc/cron.monthly*.

Cron se ocupa de las tareas rutinarias diariamente a las 6:25 de la mañana (los scripts ejecutables en */etc/cron.dialy*), incluyendo al script *logrotate*, el cual rota, comprime y clasifica los ficheros log.

Si el ordenador que se está usando no corre veinticuatro horas al día, siete días a la semana, se aconseja modificar esas entradas y especificar las horas en las que se sabe que la máquina estará activa:

```
25 17  * * *  root  ➥
test -x ➥
/usr/sbin/anacron || ➥
run-parts --report ➥
/etc/cron.daily
```

Asistentes Cron Basados en GUI

Existen gran cantidad de herramientas basadas en GUI que ayudan a crear una tabla cron. Los usuarios de Gnome tienen Gcrontab, un programa muy bien organizado y de fácil uso que permite confeccionar una programación con unos cuantos clics de ratón. La herramienta KCron de KDE es incluso más fácil de usar, sin embargo, no presenta las entradas que crea una sintaxis cron, lo cual no ayuda a familiarizarse con cron (Figura 1). Pero, como suele ocurrir, la línea de comandos proporciona una mayor flexibilidad y potencia, y siempre será más rápido escribiendo entradas que hacer clic en un montón de botones. ■

Primeros pasos en la programación en Bash

SCRIPTS A MEDIDA

Con un par de trucos y algo de programación, podremos ahorrar tiempo automatizando esas tareas que se repiten una y otra vez.

POR AELEEN FRISCH

Los scripts de shell pueden convertirse en los mejores aliados de los perezosos. Y esto a pesar de que escribir código es algo que se supone que requiere tesfuerzo, pero es cierto. Es verdad que escribir un programa para el shell requiere invertir algo de tiempo, pero una vez terminado, su uso nos liberará de todo el tiempo que asignamos a esa tarea. En este artículo trataremos de explicar la programación Bash a través de su uso en ciertas situaciones comunes. Debido al carácter introductorio del artículo, ciertos detalles serán pasados por alto, pero daremos suficiente información para que puedas empezar a desarrollar tus propios scripts.

Hola, Bash

En su forma más simple, un script shell no es más que un fichero que contiene una secuencia de instrucciones a ejecutar. Por ejemplo, un usuario podría escribir el siguiente script para evitar tener que escribir todo el comando *tar* cada vez que desea hacer una copia de seguridad de sus fotos digitales:

```
#!/bin/bash
tar cvzf /save/pix.tgz ➥
/home/chavez/pix /graphics/rdc ➥
/new/pic/rachel
```

El script debe comenzar con una línea que lo identifique como tal. El par de caracteres #!, comúnmente conocidos como "shebang", deben tener a continuación la ruta completa al programa shell que los puede interpretar: en este caso, el ejecutable corresponde al shell Bash. El resto del script corresponde al comando *tar* a ejecutar.

Sólo nos queda hacer una cosa más antes de poder utilizar el script. El usuario debe dar permiso de ejecución al fichero para que el shell sepa que es un script ejecutable. Si el fichero se llama *mytar*, el siguiente comando *chmod* hará precisamente esto (suponiendo que el fichero se encuentra en el directorio actual):

```
$ chmod u+x mytar
$ ./mytar
```

El segundo comando es el que ejecuta el script, tras lo cual empezará a mostrar los mensajes de salida propios del comando *tar*.

De esta forma el usuario ha conseguido reducir el trabajo que requiere crear un archivo *tar*, y en vez de escribir los 75 caracteres del comando completo, ya sólo tiene que escribir 8. Podríamos hacer el script algo más general – y potencialmente más útil – si permitimos que el usuario pueda pasar como parámetros desde la línea de comandos, las rutas a los directorios donde se encuentran los ficheros a los que deseamos hacer copia de seguridad:

```
$ ./mytar /home/chavez ➥
/new/pix/rachel /jobs/proj5
```

Este comando hace un backup de otros ficheros diferentes al del script anterior. En el Listado 1 podemos ver la versión modificada del programa, que presenta nuevas características:

- El comando *tar* redirige la salida estándar para eliminar todos los mensajes, menos los que son de error.
- El comando *tar* se ejecuta de forma condicional, ya que se encuentra dentro de una sentencia *if*. Si la condición indicada entre corchetes es evaluada a *true*, se ejecutarán las instrucciones incluidas dentro de la sentencia, en otro caso son ignoradas.
- La condición del *if* comprueba que el número de argumentos pasados al script, (y almacenado en la variable de sistema *$#*) sea mayor que 0. Si es así, entonces quiere decir que el usuario ha indicado la lista de elementos sobre los que quiere hacer la copia de seguridad. Pero si vale 0, entonces es que el script ha sido ejecutado sin argumentos, por lo que no hay nada que hacer, y el comando *tar* no será ejecutado.
- Los parámetros pasados desde la línea de comandoa son añadidos al programa *tar* usando la variable de shell *$@*, que contiene la lista completa de argumentos. Por tanto, para el ejemplo anterior, la llamada al script se transformará en:

```
tar czf /save/mystuff.tgz ➥
/home/chavez /new/pix/rachel ➥
/jobs/proj5 > /dev/null
```

Los parámetros pasados desde la línea de comandos son pasados directamente al programa *tar*, lo cual permite al programa hacer las copias de seguridad de todos los ficheros requeridos.

Fichero de Entrada

La siguiente versión del script (ver Listado 2) cambia algo la forma de ejecución. Ahora el primer argumento se supone que será el nombre del fichero que contiene la lista de directorios a los que hacer el backup. Si hay más parámetros, éstos serán considerados como directorios adicionales a los que realizar copia de seguridad.

El script hace uso de dos variables internas: *DIRS* y *OUTFILE*. Seguiremos la convención de escribir los nombres de variables en mayúsculas para simplificar su identificación, aunque no sea obligatorio. La primera sentencia del programa asigna a la variable *DIRS* el contenido del fichero pasado como primer parámetro al script. Esto se consigue capturando la salida del comando *cat* mediante el uso de las comillas invertidas (`).

Las comillas invertidas ejecutan el comando incluido entre ellas, y son reempla-

Listado 1: Fichero de Backup Modificado

```
01 #!/bin/bash
02 # Se asegura de que al menos hay un
   argumento
03 if [ $# -gt 0 ]; then
04 tar czf /save/mystuff.tgz $@ >
   /dev/null
05 fi
```

zadas por la salida del comando ya ejecutado. Aquí, el programa *cat* leerá el contenido del fichero indicado como primer parámetro del script (que corresponde con la lista de directorios) con la intención de sacarlo por la salida estándar. Las comillas invertidas capturarán esta salida y la colocarán entre las comillas dobles de la sentencia de asignación, creando durante este proceso la variable *DIRS*. Observe que los retornos de carro en la lista de directorios no afectan a la asignación.

Una vez leído el fichero, habremos terminado con el primer parámetro, con lo que podremos usar el comando *shift* para borrarlo de la lista de argumentos. La nueva lista de parámetros estará constituida por la lista de directorios adicionales que se pasaron como argumentos desde la línea de comandos. Esta lista modificada es accesible desde la variable *$@*. De esta forma, el usuario del script puede tener una lista de directorios fijos a los que hacer el backup, y añadir nuevos directorios desde la línea de comandos cuando lo necesite.

La tercera instrucción define la variable *OUTFILE* usando la salida del comando *date*. La sintaxis utilizada es una variante de las comillas invertidas: *`command`* es equivalente a *$(command)*. Este tipo de instrucción se conoce como "sustitución de comandos". La última sentencia ejecuta *tar*, pasando como parámetro los directorios extraídos del fichero indicado como primer parámetro del script, al que se le añaden el resto de los directorios a los que se les quiere hacer backup, pasados desde la línea de comandos. Observe que se utiliza el signo dólar como prefijo cada vez que se desea recuperar el valor de una variable: *$DIRS*.

Añadiendo Comprobaciones

El Listado 2 no tiene tanto cuidado a la hora de comprobar la validez de los argumentos pasados a través de la línea de comandos. La versión del Listado 3 es algo más sofisticada e incorpora algunas de las verificaciones que ya se hacían en la primera versión del programa,

Listado 2: Estableciendo el Fichero de Entrada

```
01 #!/bin/bash
02 DIRS="`cat $1`" # Almacena el
   fichero del 1º argumento
03 shift          # Elimina el 1º argu-
   mento de la lista de parámetros
04 OUTFULE="$( date +%y%m%d )" # El
   nombre del fichero contiene la
   fecha
05 tar czf /tmp/$OUTFILE.tgz $DIRS
   $@ > /dev/null
```

además de proporcionar algo más de flexibilidad. Esta versión usa el comando interno de Bash *getopts* para un procesado simple de la línea de comandos.

Las dos primeras instrucciones asignan valores a las variables *DEST* y *PREFIX*. La primera variable almacena la ruta del directorio donde quedará alojado el fichero *tar*, mientras que la segunda guarda un prefijo para el nombre del fichero. A éste se le añadirá al final una cadena con la fecha de creación. El resto del programa se estructura bajo un bucle *while*:

```
while condition-cmd;
commands
done
```

El bucle se ejecuta mientras se cumpla la condición y termina una vez ésta sea evaluada a falso. Aquí la condición a evaluar es *getopts "f:bn:d:" OPT*. En Bash, las expresiones condicionales se encierran entre corchetes (como ya vimos en la anterior sentencia *if*), pero si en vez de una expresión condicional queremos evaluar la salida de un comando, éste no deberá estar encerrado entre corchetes. Técnicamente los corchetes constituyen en sí mismo un comando: ejecutan el programa *test*. Un programa es evaluado a verdadero mientras devuelva alguna salida, y a falso cuando su salida se agote.

La función *getopts* nos devuelve los parámetros usados desde la línea de comandos junto a sus argumentos. En vez de devolver-

Listado 3: Check de Restauración

```
01 #!/bin/bash
02 DEST="/save"   # Directorio por
   defecto donde almacenamos el
   backup
03 PREFIX="backup" # Prefijo del
   fichero
04 while getopts "f:bn:d:" OPT; do #
   Analizamos los argumentos
05 case $OPT in
06 n) PREFIX=$OPTARG ;; # -n <pre-
   fix>
07 b) ZIP="j"; EXT="tbz" ;; # -b =
   use bzip2 not gzip
08 f) DIRS=$OPTARG ;; # -f
   <dir-list-file>
09 d) if [ "${OPTARG:0:1}" = "/" ]
   # -d <archive-dir>
10 then
11 DEST=$OPTARG
12 else
13 echo "Destination directory must
   begin with /."
14 exit 1 # acabar con estado de
   error
15 fi
16 ;;
17 :) echo "You need to give an argu-
   ment for option \u2010$OPTARG."
18 exit 1
19 ;;
20 *) echo "Invalid argument:
   \u2010$OPTARG."
21 exit 1
22 ;;
23 esac
24 done
25 if [ -z $DIRS ]; then # Nos asegu-
   ramos de tener un elemento válido
26 echo "The -f list-file option is
   required."
27 exit 1
28 elif [ ! -r $DIRS ]; then
29 echo "Cannot find or read file
   $DIRS."
30 exit 1
31 fi
32 DAT="$( /bin/date +%d%m%g )"
33 /bin/tar -${ZIP-z} -c -f
   /$DEST/${PREFIX}_$DAT.${EXT-tgz
   } `cat $DIRS` > /dev/null
```

Listado 4: Añadiendo una Condición Numérica

```
01 #!/bin/bash
02 if [ $# -lt 1 ]; then    # Si no se
   pasa ningún argumento, pregunta
03 read -p "Who did you want to check
   for? " WHO
04 if [ -z $WHO ]; then    # No se
   introduce ningún nombre
05 exit 0
06 fi
07 else
08 WHO="$1"              # Guardamos el
   argumento
09 fi
10
11 LOOK=$(w | grep "^$WHO")
12 if [ $? -eq 0 ]; then    # Miramos el
   estado previo
13 WHEN=$(echo $LOOK | awk '{print
   $4}')
14 echo "$WHO has been logged in
   since $WHEN."
15 else
16 echo "$WHO is not currently logged
   in."
17 fi
18 exit 0
```

los directamente, *getopts* asigna a la variable indicada como segundo parámetro – *OPT* en el ejemplo – la letra usada para el argumento, mientras que su valor lo asigna siempre a la variable *OPTARG*. El primer argumento de *getopts* es una cadena que lista todas las letras válidas que se esperan como parámetros (diferencia entre mayúsculas y minúsculas). Las letras que preceden a los dos puntos requieren un argumento – en este caso son *f*, *n* y *d*. Todas estas letras (demanden o no un argumento adicional) deben tener un guión delante para poder ser usadas como parámetros desde la línea de comandos.

Dentro del bucle *while* hay una sentencia *case*. Esa sentencia siempre compara el valor de la expresión indicada como su argumento – en este caso el valor de la variable *OPT* establecida por *getopts* – con una serie de patrones alojados bajo ella. Cada patrón consta de una cadena (quizás con comodines) terminada por un paréntesis de cierre. El orden en el que se escriben estos patrones es importante, puesto que en caso de empate, el primero que case con la expresión evaluada, gana.

En este ejemplo, los patrones son las propias letras que conforman un parámetro válido, los dos puntos y el comodín asterisco, que casará con cualquier otra posible condición no especificada anteriormente (es decir, cualquier otra que no sea *n*, *b*, *f*, *d* o :). Cada opción es procesada por un conjunto diferente de instrucciones, y cada sección termina con dos puntos y coma (;;). Desde el código se puede ver que el parámetro *-n* se utiliza para establecer el prefijo usado para el nombre del archivo (sobrescribe el valor por defecto establecido en la segunda línea del programa), *-b* se usa para indicar una compresión bzip2 en vez de gzip (como veremos después), con *-f* indicamos el fichero que contiene la lista de elementos a los que se les realizará el backup, y *-d* el directorio de destino del archivo *tar* (que por defecto está a */save* tal y como se establece en la primera instrucción del script).

Una de las cosas que se comprueba es que el directorio de destino establecido por *-d* contiene la ruta absoluta. Para ello se usa la instrucción *${OPTARG:0:1}*. La sintaxis general utilizada para la sustitución de variables usando $ hace uso de las llaves alrededor del elemento referenciado. Es decir, *$1* puede escribirse como *${1}*, y *$CAT* como *${CAT}*. La utilidad de esta sintaxis está en la capacidad de acceder a los parámetros posicionales superiores a nueve; por ejemplo, ${11} hace referencia al parámetro número 11 del programa, mientras que $11 se expande al primer parámetro del script, seguido por un 1: ${1}1. Esta sintaxis posibilita además que las variables queden aisladas del texto que las rodea: si el valor de la variable *ANIMAL* es *cat*, entonces *${ANIMAL}2* es expandida a *cat2*, mientras que *$ANIMAL2* haría referencia al valor de una variable llamada *ANIMAL2*, que probablemente esté sin definir. Observe que los puntos no forman parte de los nombres de variables (como se mostrará más adelante).

El sufijo *:0:1* que sigue al nombre de la variable se utiliza para extraer de *OPTARG* la subcadena que comienza en la primera posición (los caracteres se enumeran comenzando desde 0), y continúa 1 carácter más. Es decir, este sufijo se utiliza para conseguir el primer carácter de la variable. El comando *if* mira a ver si ese carácter corresponde a la barra de directorio, si no es así muestra un mensaje de error y sale del programa devolviendo un valor de estado 1. Un valor de estado 1 indica que el programa ha terminado con algún tipo de error, mientras que el código de estado 0 se usa para salida de programas que han terminado con éxito.

Si a un parámetro que requiere un argumento no se le pasa ninguno desde la línea de comandos, *getopts* asignará a la variable *OPT* el carácter de dos puntos (:) y a la variable *OPTARGS* la cadena que representa a ese parámetro. La penúltima sección de la sentencia *case* gestiona este tipo de error. La última sección es la que se encarga de procesar cualquier parámetro inválido encontrado. Cuando este es el caso, *getopts* asigna a *OPTS* el carácter de interrogación, y a *OPTARGS* la cadena que representa la opción desconocida; el comodín casará con cualquier opción no contemplada para así poder gestionarla.

Esta forma de gestionar los parámetros no es todo lo robusta que podría ser. Algunas combinaciones de parámetros no válidas no se detectan hasta el final del script (por ejemplo, si pasamos los parámetros *-f -n*, el argumento de *-f* no viene dado, con lo que *-n* es interpretado incorrectamente como tal).

La sentencia *if* analiza dos posibles problemas que podemos tener con el fichero que

Listado 5: Informando sobre el Espacio en Disco

```
01 #!/bin/bash
02 PATH=/bin:/usr/bin # Establece
   el PATH
03 . /usr/local/sbin/function.bash
   # . f => incluye el fichero f aquí
04 printf "USER\tGB USED\n" #
   Imprime la cabecera del informe
05 for WHO in
   $(</usr/local/sbin/ckusers); do
06 HOMESUM=`eval du -s ~$WHO | awk
   '{print $1}'`
07 TMPLIST=$( ls -lR --block-size
   1024 $(</usr/local/bin/ckdirs) |
   egrep "^.......... +[0-9]+ $WHO"
   | awk '{print $5}')
09 TSUM=0
10 for N in $TMPLIST; do
11 TSUM=$(( $TSUM+$N ))
12 done
13 TOT=$(( $HOMESUM+$TSUM ))
14 tp_gb $WHO $TOT
15 done
```

contiene la lista de directorios. Primero mira a ver si la variable *DIRS* no está definida. Si este es el caso, la variable tendrá longitud cero, y el programa saldrá con un mensaje de error. La segunda prueba se realiza dentro de *elif* (contracción de *else if*). Ésta se asegura de que el fichero existe y que tiene permisos de lectura. Si no es el caso (el operador *!* hace de NO lógico), el script mostrará un mensaje de error antes de salir.

Los dos últimas instrucciones del programa son las que, por un lado, componen el prefijo del nombre del fichero de copia de seguridad a partir de la salida del comando *date*, y por otro, ejecutan el comando *tar*. El comando *tar* hace uso de variables condicionalmente referenciadas – por ejemplo, *${EXT-tgz}*. El guión (-) que sigue a la variable indica que si la variable no está definida, se deberá usar la cadena que la sigue. Las variables *EXT* y *ZIP* sólo las vamos a tener definidas si el usuario hace uso del parámetro de línea de comandos *-b*. En ese caso se asignarán a las variables los valores *tbz* y *j* respectivamente. Si no, se usarán los valores *tgz* y *z*.

Condiciones Numéricas

Hemos puesto ejemplos de condiciones basadas en cadenas y en propiedades de ficheros. El Listado 4 introduce una condición de tipo numérico. El programa está pensado para ser usado por la secretaria del presidente de una empresa que desea comprobar cuándo ha accedido alguien al sistema.

Lo primero que hace el programa es comprobar que se ha pasado algún parámetro por la línea de comandos. Si no es así, es decir, si el número de argumentos es menor que 1, entonces preguntará por el nombre de un usuario del sistema usando el comando *read*. La respuesta se almacena en la variable *WHO*. Dentro del mismo caso, si *WHO* tiene longitud cero, es que el usuario al final no ha llegado a introducir ningún nombre de usuario y que simplemente ha pulsado la tecla de retorno de carro, con lo que el programa deberá terminar. Por otro lado, si se ha especificado un argumento desde la línea de comandos, entonces el script asignará ese valor a la variable *WHO*. En cualquier caso, la variable *WHO* contendrá el nombre del usuario que buscamos.

La segunda parte del programa nos enseña a usar dos variantes de sustitución de coman-

Tabla 1: Resumen de los Comandos Bash

Argumentos y Variables	
$1 ... $9	Argumentos
${nn}	Forma general para el argumento *nn*
$@	Todos los argumentos en una lista de elemento
*$**	Todos los argumentos, pero en un solo ele mento
$#	Número de argumentos
$0	Nombre del Script
$var	Valor de la variable *var*
${var}	Forma general
${var:p:n}	Subcadena de *n* caracteres, comenzando en *p*
${var-val2}	Devuelve *val2* si *var* no está definida
${var+val2}	Devuelve *val2* si *var* está definida
${var=val2}	Devuelve *val2* si *var* no está definida, y establece *var=val2*
${var?errmsg}	Muestra "var: errmsg" si *var* no está definida
arr=(items)	Define *arr* como array
${arr[n]}	Elemento *n* del array *arr*
${#arr[@]}	Número de elementos definidos en *arr*
getopts opts var	Decodifica los argumentos posicionales de un script. *opts* contiene las letras de opción que debe reconocer, y si éstas aceptan parámetros o no. Cada invocación a *getopts* procesa un solo argumento y asigna el nombre del parámetros en *var* y el valor (si existe) en *$OPTARG*
Comandos Generales	
`cmd	Sustituye la salida de *cmd*
$(cmd)	Sustituye la salida de *cmd*
$?	Valor de la salida del último comando
$!	PID del último comando ejecutado en background
eval string	Evalúa la cadena tras la sustitución de variables
. file	Incluye el contenido del fichero dentro del script
exit n	Sale del script devolviendo un valor (0 significa éxito)
Aritmética	
$((expresion))	Evalúa la expresión como operación entre enteros
*+ - * /*	Suma, resta, multiplicación y división entera
++ --	Incremento y decremento
%	Módulo
***	Exponenciación

Condiciones	
-x file	Mira si la condición '-x' se cumple en el fichero. Algunas condiciones pueden ser: '-s' longitud mayor que 0; '-r' permiso de lectura; '-w' permiso de escritura; '-e' el fichero existe; '-d' es un directorio; '-f' es un fichero normal
file1 -nt file2	El fichero 1 se ha creado antes que el 2
-z string	La longitud de la cadena es 0
-n string	La longitud de la cadena es mayor que 0
string1 = string2	Las dos cadenas son iguales. Otros operadores: !=, >, <
int1 -eq int2	Los dos enteros son iguales. Otros operadores: -ne, -gt, -lt, -ge, -le
!	NOT
-a	AND
-o	OR
( )	Agrupa condiciones
Entrada y Salida	
read vars	Lee una línea desde la entrada estándar y asigna una palabra a cada una de las variables
read -p string vars	Lee una línea desde la entrada estándar, pero muestra antes un prompt
printf fstring vars	Muestra los valores de *vars* según el formato especificado en *fstring*. La cadena de formato puede contener caracteres especiales como '\n', '\t', texto normal y códigos de formato. Algunos de los códigos más usados son: '%s' para imprimir cadenas, '%d' para números enteros con signo o '%f' para números flotantes. Se pueden añadir modificadores de formato, como un signo '-' entre el porcentaje y el código de formato para indicar alineamiento a la derecha. El código de formato puede estar precedido por un número que indicará en ancho del campo. Por ejemplo, '%-5d' indicará que se mostrará un número entero de cinco dígitos alineados a la derecha, '%6.2f' mostrará un número flotante de 6 dígitos, donde dos de ellos serán decimales
Funciones	
name(){comandos}	Usa *local* para limitar el alcance de las variables declaradas dentro de la función

dos. La primera variante busca el nombre del usuario dentro de la salida del programa *w*, asignando a la variable *LOOK* la línea de interés en caso de que la encontremos. La segunda forma define la variable *WHEN* como el cuarto campo de esa línea capturada en el paso anterior. Este campo corresponde a la hora de entrada más reciente del usuario, y es extraído usando el programa *awk*. Por suerte, no es mucho lo que tenemos que aprender de *awk* para poder extraer campos que se encuentran separados por espacios.

Este programa sólo se ejecuta si *$?* es igual a 0. Bash almacena en *$?* el código de estado devuelto por el último programa ejecutado: *grep*. *grep* devuelve 0 sólo cuando ha encontrado alguna coincidencia, y 1 en otro caso. Al final el programa mostrará un mensaje descriptivo donde nos devuelve el estado del usuario:

```
kyrre has been logged in since 08:47.
```

While y Read

La siguiente secuencia de comandos nos enseña otro uso para *while* y *read*. Estas instrucciones pueden ser usadas para el procesamiento de la salida estándar o el contenido de un fichero. El objetivo del script es el de enviar mensajes por email a una lista de usuarios:

```
#!/bin/bash
/bin/cat ➤
/usr/local/sbin/email_list |
while read WHO WHAT SUBJ; do
/usr/bin/mail -s ➤
"$SUBJ" $WHO < $WHAT
echo $WHO
done
```

El script envía el contenido del fichero *email_list* al comando *while*; la salida del comando *read* se convierte en la condición usada en el *while*. El programa *read* lee una a una todas las líneas de la entrada estándar – que corresponde a la salida del comando *cat* – y asigna la primera palabra a la variable *WHO*, la segunda palabra a *WHAT*, y el resto de las palabras de la línea a *SUBJ* (por defecto se supone que las palabras están separadas por espacios). Estas variables se utilizan para guardar la dirección de email, el nombre del fichero que contiene el mensaje de texto y una cadena con el asunto del correo. Estas variables son usadas para la construcción de la llamada al comando *mail*.

Observe que en este ejemplo se han usado rutas absolutas para indicar la localización de los programas externos. Esto debería ser práctica habitual, aunque también sería una solución válida el incluir un *PATH* explícito al comienzo del programa. Esto nos evitaría los problemas de seguridad derivados de la sustitución de ejecutables en el sistema. Desafortunadamente el script es excesivamente confiado respecto al contenido del fichero *email_list*. No se asegura en ningún momento de que las direcciones de correo estén adecuadamente formateadas. Cualquier futuro uso del programa va a requerir de un cuidadoso análisis manual de las direcciones de email incluidas en el fichero. Imagine lo que podría suceder si existieran líneas como *jane@ahania.com; /somewhere/run_me* en la lista de direcciones.

Bucles

Los siguientes dos programas muestran otro tipo de bucle usado comúnmente en la programación de scripts shell: el comando *for*. El programa del Listado 5 tiene como objetivo el preparar un informe de la cantidad total de espacio en disco que ocupan los ficheros alojados en una lista de directorios, y que pertenecen a un conjunto de usuarios dados. Los ficheros que contienen la lista de usuarios y directorios a examinar se encuentran fijados en el código. El programa comienza fijando el *PATH* e incluyendo el contenido de otro script dentro de éste. Para ello hace uso del comando punto como mecanismo para la inclusión de ficheros.

Dentro del programa hay una serie de elementos a destacar:

- El comando *for* hace uso de una variable, la palabra clave *in*, una lista de elementos, y finalmente el comando *do*. En cada iteración del bucle (que termina con *done*), a la variable se le asigna el siguiente elemento de la lista. Es decir, a *WHO* se le asignan cada una de las líneas del fichero *ckusers*. La instrucción *$(< file)* es una versión corta de *$(cat file)*.
- La definición de *HOMESUM* hace uso de las comillas inversas para extraer el tamaño total del directorio de usuario de la salida del comando *du -s* (para luego extraer el campo de interés con *awk*). El comando *eval* hace que *du* pueda interpretar la versión ya expandida de *~$WHO* como un directorio de usuario que hace uso de la nomenclatura de la tilde.
- La definición de *TMPLIST* usa el comando de sustitución para almacenar el campo que contiene el tamaño (otra vez con ayuda de *awk*) de todas las líneas de *ls -lR*. Con *egrep* filtramos las líneas de salida de *ls* para quedarnos con aquellas que corresponden a ficheros cuyo dueño es el usuario actual. El comando *ls* se ejecuta sobre los directorios indicados en el fichero *ckdirs* y hace uso del parámetro *--block-size* para obtener el tamaño de fichero en unidades compatibles con *du* (KB). *TMPLIST* es una lista de números: uno por cada fichero del que es dueño el usuario actual (*$WHO*).
- El segundo bucle *for* suma los números de *TMPLIST* y asigna el resultado *TSUM*. La variable *N* es usada durante la iteración, y toma los valores de los elementos de la lista *TMPLIST*.
- El shell Bash proporciona aritmética con enteros por medio del comando *$((math-expression))*. El script hace uso de esta forma dos veces.
- El programa usa la función *to_gb* para imprimir cada línea del informe. Bash requiere que las funciones estén definidas antes de poder ser usadas, así que éstas son normalmente alojadas en ficheros externos e invocadas por medio del mecanismo de inclusión de ficheros del comando punto. La función se encuentra en el fichero *functions.bash*.

La función *to_gb* se muestra en el Listado 6. Comienza definiendo algunas variables

Listado 6: to_gb

```
01 to_gb()
02 {
03 # Argumentos: user usage-in-KB
04 local MB D1 D2 USER    # Variable
   locales
05 USER=$1
06 MB=$(( $2/1024 ))     # Conversión
   aprox. a MB
07 D1=$(( $MB/1000 ))    # Nos
   quedamos con la parte entera (GB)
08 D2=$(( $MB-($D1*1000) )) # Calcu-
   lamos el resto
09 # Mostramos "XXXX MB" como: "X.XXX
   GB"
10 printf "%s\t%s\n" $USER
   $D1.${D2:0:1}
11 return
13 }
```

Listado 7: Script Factorial

```
01 #!/bin/bash
03 F=1
04 for (( I=$1 ; I>1 ; I-- )); do
05 F=$(( $F*$I ))
06 done
07 echo $1'! ='$F
08 exit 0
```

locales. La función ignorará aquellas que coinciden en nombre y se encuentran dentro del script llamante, y los nuevos valores no se retornarán a ese mismo script. El grueso de la función consiste en un montón de operaciones aritméticas que usan *$((...))*. Bash proporciona sólo aritmética entera, pero deseamos proporcionar información precisa del tamaño total en gigabytes. Para ello usaremos un pequeño truco, muy conocido, que nos permite extraer la parte entera y el resto de un valor expresado en gigabytes y construir el proceso de impresión de forma manual. Por ejemplo, si tengo 2987MB y lo dividimos por 1024, nos daría 2GB. En vez de esto, dividimos 2987 por 1000 (*D1 = 2*) para luego calcular 2987 - (2*1000) (*D2 = 987*). Luego podremos imprimir *D1*, un punto decimal y el primer carácter de *D2*: *2.9*.

El comando *printf* se usa para la generación de salida formateada. Requiere de una cadena de formato seguida de las variables que van a ser mostradas. El contenido dentro de la cadena de formato es indicado mediante un código de letras tras el signo porcentual. Dentro de la cadena de formato podemos usar los caracteres *\t* y *\n* para indicar tabulador y retorno de carro respectivamente. Este último carácter debe ser incluido de forma explícita cada vez que deseemos terminar una línea.

Alguna salida de ejemplo de este script:

```
USER   GB USED
seleen 80.5
kyrre  14.3
munin  0.3
```

El siguiente programa (Listado 7), que calcula factoriales, muestra un tipo de bucle *for* similar al que encontramos en muchos lenguajes de programación (la sintaxis es parecida a C).

La sintaxis para *for* proporciona una variable para el bucle, junto con su valor de inicio, una condición para la continuidad del bucle, y una expresión que indica cómo la variable debe ser modificada tras cada iteración. Aquí el bucle hace uso de la variable *I*, cuyo valor de inicio es el que el usuario asigne como primer parámetro del script. Al final de cada iteración el valor es decrementado en 1 (*I++* incrementará I), y el bucle continúa mientras I sea mayor que 1. El cuerpo del bucle multiplica *F* (que estaba inicializado a 1) por cada sucesivo valor de *I*. El script termina imprimiendo el resultado final:

```
$ ./fact 6
6! = 720
```

Generación de Menús

El último script muestra la capacidad interna de Bash para generar menús mediante el comando *select* (Listado 8). La preparación del comando *select* sucede durante la definición de las variables *PKGS* y *MENU*. El comando *select* requiere de una lista de elementos como segundo argumento, y *MENU* servirá para ese propósito. Queda definido a través de una construcción de comando de sustitución. Aquí añadimos la cadena literal *Done* al final de la lista.

La definición de *PKGS* introduce una nueva característica: los arrays. Un array es una estructura de datos que contiene múltiples elementos que pueden ser referenciados mediante un índice. El siguiente ejemplo define y usa un simple array:

```
$ a=(1 2 3 4 5)
$ echo ${a[2]}
3
```

Un array queda definido encerrando sus elementos entre paréntesis. Los elementos del array pueden ser referenciados usando la sintaxis de la segunda línea: el nombre del array dentro de las llaves y el elemento deseado especificado dentro de los corchetes. Observe que la numeración de los elementos comienza en 0. Bajo circunstancias normales, el número de elementos de un array viene dado por *${#a[@]}*. *PKGS* es definido como un array que se construye a partir del segundo campo de cada línea del fichero.

El comando *select* usa el contenido de *MENU* como su lista. Generará un menú de texto enumerado a partir de la lista de elementos y preguntará al usuario por una opción. El elemento seleccionado es devuelto en la variable indicada tras *in* (aquí *WHAT*), y el número de elemento es devuelto en la variable *REPLY*.

El script usará el valor de *REPLY* menos 1 para recuperar el nombre de paquete correspondiente del array *PKGS* de la variable *PICKED* (usamos *$REPLY-1*, puesto que la numeración del menú comienza desde 1, pero los elementos del array comienzan en 0). El comando *select* sale cuando el usuario selecciona el elemento *Done*.

A continuación se muestra un ejemplo de ejecución del script:

```
1) CD/MP3_Player  3) Photo_Albun
2) Spider_Solitare 4) Done
#? 2
Installing package spider ...
Please be patient!
many more messages ...
#? 4
```

Listado 8: Generación de Menús

```
01 #!/bin/bash
02 PATH=/bin:/usr/bin
03 PFILE=/usr/local/bin/userpkgs
   # Formato: pkgname menu_item
05 PKGS=( $(cat $PFILE | awk '{print
   $1}') ) # Array de nombre de paque-
   tes
06 MENU="$( cat $PFILE | awk '{print
   $2}') Done" # Listado de los ele-
   mentos del menú
07 select WHAT in $MENU; do
08 if [ $WHAT = "Done" ]; then exit;
   fi
09 I=$(( $REPLY-1 ))
10 PICKED=${PKGS[$I]}
11 echo Installing package $PICKED
   ... Please be patient!
12 # ... comandos para la instalación
   del paquete
13 done
```

Conclusiones

La tabla "Resumen de los Comandos Bash" contiene una referencia rápida de algunos comandos de script de Bash. Esperamos que haya disfrutado de esta pequeña incursión en el mundo de la programación en Bash. Estas técnicas pueden ser usadas para construir sus propios scripts Bash para la automatización de ciertas tareas rutinarias. Esperamos que se lo pase bien en las siguientes exploraciones. ■

Perfeccionamiento de scripts en Bash

PROGRAMANDO SCRIPTS

Con todas las novedades añadidas a las últimas versiones de Bash, rara vez hace falta el apoyo de programas externos. **POR MIRKO DÖLLE**

Muchos scripts tratan con Bash como si sólo fuera capaz de poco más que llamar a programas externos. Esto es sorprendente, desde la versión 2 de la shell por defecto hay disponible un conjunto de comandos que abarcan de todo, desde la manipulación compleja de las cadenas de caracteres, pasando por las expresiones regulares, hasta los arrays; lo que hace que sean redundantes las llamadas a los programas externos.

La principal ventaja de las funciones internas es que la shell no tiene que lanzar un proceso nuevo, lo que ahorra tiempo de proceso y memoria. Esta capacidad puede ser importante, particularmente si hace falta ejecutar un programa como *grep* o *cut* en un bucle, ya que el consumo de tiempo y memoria del script puede dispararse si no se tiene cuidado. Este artículo describe algunas técnicas simples para acelerar los scripts Bash.

En el Banco de Pruebas

Los siguientes scripts evalúan un fichero de registro de Apache de un sitio web para realizar una comparativa. Si está interesado en conocer qué páginas han sido solicitadas, debe aislar la cadena GET del fichero de registro de la siguiente manera:

```
84.57.16.30 - - ➥
[21/Oct/2005:04:18:26 +0200] ➥
"GET /favicon.ico HTTP/1.1" ➥
404 209 "-" "Mozilla/5.0 ➥
(X11; U; Linux i686; de-DE; ➥
rv:1.7.5) Gecko/20041122 ➥
Firefox/1.0"
```

El Listado 1 muestra una solución que utilizan muchos scripts Bash. La llamada a *cat* en la Línea 3 lee primero el fichero de registro entero, y trabaja con él en el bucle *for* como si se tratara de una lista de parámetros, con lo que Bash tiene que mantener en la caché el contenido del primer fichero. En la Línea 5, Listado 1, se continúa realizando una llamada al programa externo *cut* por cada línea del fichero de registro. La ejecución del script en una máquina Pentium III a 750Mhz tardó 18,5 segundos en el análisis de un fichero de registro de 600Kbyte de Apache.

Por otro lado, el script del Listado 2 tan sólo tardó 3.3 segundos, casi seis veces más rápido: utiliza el descriptor de ficheros 3 para abrir el fichero y hace uso de la variable *Request* para procesar una línea en cada vuelta del bucle, con lo que Bash sólo necesitará mantener en la caché una única línea del fichero de registro. A continuación, el script elimina los caracteres del comienzo de la línea hasta que aparece GET, incluido éste, y todos los caracteres del final de la línea hasta *HTTP/*, incluido.

Listado 1: Evaluación Externa de los Registros

```
01 #!/bin/bash
02 IFS=$'\n'
03 for l in `cat access.log`; do
04 IFS=" "
05 echo "${l}" | cut -d" " -f7
06 done
```

Listado 2: Evaluación Interna de los Registros

```
01 #!/bin/bash
02 exec 3<access.log
03 while read -u 3 Request; do
04 Request="${Request##*GET }"
05 echo "${Request%% HTTP/*}"
06 done
```

Listado 3: Grep Interno

```
01 #!/bin/bash
02 exec 3<$2
03 while read -u 3 line; do
04 if [ -z "${line/*${1}*}" ];
   then
05 echo "$line"
06 fi
07 done
```

Listado 4: Funciones de Cadena de Caracteres

```
01 #!/bin/bash
02 function GetIP() {
03 while read -u $1 Request; do
04 tmp="${Request%% *}"
05 IP[1]="${tmp%%.*}"
06 IP[4]="${tmp##*.}"
07 tmp="${tmp%.*}"
08 IP[3]="${tmp##*.}"
09 tmp="${tmp%.*}"
10 IP[2]="${tmp##*.}"
11 printf
   "%03d.%03d.%03d.%03d\n"
   ${IP[1]} ${IP[2]} ${IP[3]}
   ${IP[4]}
12 done
13 }
14 exec 3<access.log
15 GetIP 3 | sort | uniq
```

Se puede ahorrar otra décima de segundo eliminando todos los caracteres que van hasta el blanco del final de la petición, ya que se simplifica la comparación de la cadena por la función interna de Bash.

Funciones de Sustitución

Las herramientas *basename* y *dirname* son fácilmente sustituibles con las funciones Bash. Para realizar esta operación se necesita la función de cadena *${Variable%Patrón}*, que elimina la cadena más corta que coincida con el patrón de la parte final de la cadena y *${Variable##Patrón}*, que elimina la cadena más larga que coincida con el patrón de la parte inicial de la cadena. Con un simple *alias* se reemplaza realmente *dirname*:

```
alias dirname=echo ${1%/*};
```

La función *basename* no es mucho más compleja, aunque hay que considerar que *basename* puede eliminar la extensión del fichero, por ello hay que combinar las dos funciones de cadena:

```
function basename() {
B=${1##*/}
echo ${B%$2}
}
```

Es más eficiente almacenar los resultados de la cadena truncada en la variable B que establecer el primer parámetro y pasarle el segundo parámetro.

No tiene sentido reemplazar cada llamada a un programa con los comandos internos. El Listado 3 es un buen ejemplo de ello: implementa un *grep* rudimentario. Aunque el script comprende solamente unas cuantas líneas de código y podría parecer eficiente a simple vista, tarda unos dos minutos en buscar un nombre de fichero en los 600Kbytes del fichero de registro del servidor web (*grep* lo realiza en 0,1 segundos).

Si se encuentra una coincidencia, la línea analizada del fichero se borra completamente por una "búsqueda y reemplazo" en la Línea 4. El patrón de búsqueda, **${1}**, le indica a Bash que busque cada carácter individual de la línea hasta encontrar una coincidencia. Si se utiliza *#*${1}** como patrón de búsqueda, diciéndole a Bash que busque al comienzo de la línea únicamente, Bash solamente realiza la comparación una vez por línea reduciendo el tiempo del script a menos de tres segundos.

Diseccionando IPs

Pero existe una forma más elegante y rápida de diseccionar las cadenas en algunos escenarios. Por ejemplo, si hay que ordenar las direcciones IP por las que fueron originadas, no se puede utilizar la función *sort* o una simple función de ordenación léxica, ya que pondría a 217.83.13.152 antes que 62.104.118.59. Por el contrario, hace falta extraer los bytes individuales de la dirección IP, convertirlos a un formato ordenable, ordenarlos y por último mostrar los resultados sin los duplicados.

Los Listados 4 y 5 muestran dos soluciones posibles con características de rendimiento completamente diferentes. El script del Listado 5 comienza analizando el fichero de registro línea por línea (Línea 3) y luego extrae la primera dirección IP en la Línea 4. De la Línea 6 a la 10 se elimina un octeto cada vez de la parte final de la dirección IP y se almacena el byte de la dirección como un decimal en el array *IP*.

En la Línea 11, la llamada a *printf*, que es también un comando interno de Bash, imprime los cuatro octetos de la dirección IP separados por puntos, tres dígitos decimales con relleno de ceros. La última línea encauza la salida al programa externo *sort* antes de eliminar los duplicados. El Listado 4 tarda unos

Listado 5: Funciones de Cadena de Caracteres

```
01 #!/bin/bash
02 function GetIP() {
03 IFS=". "
04 while read -u $1 -a IP; do
05 printf
   "%03d.%03d.%03d.%03d\n"
   ${IP[0]} ${IP[1]} ${IP[2]}
   ${IP[3]}
06 done
07 }
08 exec 3<access.log
09 GetIP 3 | sort | uniq
```

2,6 segundos en procesar el fichero de registro de 600KBytes de Apache.

Funciones de Compactación

El programa del Listado 5 realiza el mismo trabajo que el del Listado 4, pero tan sólo tarda 1,6 segundos, una mejora de casi el 40 por ciento. Las funciones de la cadena de las líneas 4 a la 10 del Listado 4 son las que causan la diferencia en los tiempos de ejecución: en vez de extraer primero las direcciones IP y luego utilizar siete llamadas a funciones para diseccionarlas, el Listado 5 llama a la función interna *read* de Bash con la variable especial *IFS*. Bash trata los caracteres almacenados en *IFS* como parámetros separadores. Por defecto son caracteres espacio, tabulador y línea nueva.

La Línea 3 del Listado 5 define el punto y el espacio en blanco como separadores. Llamando a *read* con el parámetro *-a* se le indica a la función que no almacene una línea completa en la variable, sino que haga uso del separador *IFS* y que escriba los elementos de la entrada uno a uno en la variable array *IP*. Los octetos que conforman la dirección IP se almacenan en las variables que van desde la *IP[0]* hasta la *IP[3]* de la llamada a *read*. Además, una única llamada a la función en el Listado 5 reemplaza las líneas de la 3 a la 10 del Listado 4.

Se pueden reemplazar las llamadas externas a *sort* y *uniq* por funciones Bash, pero no se puede esperar que Bash sea capaz de igualar a *sort*, un programa C, en eficiencia. Como en la vida real, algunos ajustes en los scripts Bash pueden ser sorprendentes, pero programar en busca de la eficiencia también puede pagarse a la larga. ■

ÍNDICE DE COMANDOS BASH

Símbolos

. (punto) 17, 91
$0 91
$1 ... $9 91
$! 91
$# 88, 91
$(...) 89
$((...)) 91, 92
$* 91
$? 91
$@ 88, 91
${ ... } 90, 91
&& 18
< 18, 95
> 18
>> 19
[...] 89, 94, 95
`...` 89
| 19
|| 18, 19

A

alias 17, 26, 39, 95
apt-cache 62
  dump 62
  search 62
  showpkg 62
  stats 62
apt-get 60-62, 64-66
  autoclean 62
  autoremove 62
  build-dep 61
  check 62
  clean 62
  dist-upgrade 61
  install 61
  purge 61
  remove 61
  update 62
  upgrade 61
at 85, 86
atq 86
atrm 86
awk 22, 30

B

basename 95
bg 59
bookmark 79

C

cal 38
cat 20, 29, 30
case 89
cd 10, 11, 18
chgrp 47, 49
chmod 47, 48, 49, 88
chown 45, 49
cp 10, 12, 15, 18, 45
crontab 86
cron 86

D

date 39, 40
dd 68, 69
debugfs 37
/dev 41
df 6, 36
dhclient 82
dig 71
dnsdomainname 8
do 30, 89, 93, 94, 95
done 30, 89, 93, 94, 95
dpkg 61, 64
dpkg-query 61
du 36

E

echo 11, 30, 31, 90, 92, 93, 94
edquota 54
elif 88
esac 89
eth0 71
eval 91
exec 94
exit 90, 91
export 21, 25, 27, 86

F

fdisk 7, 32, 33
fg 59
find 14, 15, 19, 30, 49
fi 88, 94
for 30, 92, 93, 94
free 7, 57
fsck 37
  e2fsck 37
  fsck.vfat 37
  reiserfsck 37
ftp 77
function 95

G

gawk 23, 24
genisoimage 69
get 79
getopts 91
grep 16, 17, 90
groups 49
grpquota 53

H

head 20
help 78
history 27
$HOME 45, 74
host 71
hwclock 40

I

if 88, 94
ifconfig 70, 71
iwconfig 82
iwevent 84
iwgetid 84
iwlist 84
iwspy 84

J

jobs 59

K

killall 58
kill 17, 58

L

less 19, 65
locate 15
lo 70

lsmod 6
lsof 43
lspci 8, 29
ls 19, 48

M

make 18
mget 78
mkdir 10, 11, 15, 18, 45
mkfs 35
modinfo 28
modprobe 28
more 20
mount 41, 54, 69, 74
mtr 72
mv 10

N

ncftp 78
netstat 72
nice 57
nohup 59
ntpdate 40
ntp 40

O

open 77

P

parted 7, 32, 33
passwd 48
pgrep 59
ping 71, 84
pkill 59
printf 91, 95
/proc 8, 29
prompt 26
pstree 56
ps 17, 43, 56, 58
$PS1 27

Q

quotacheck 54
quotaoff 55
quotaon 55
quota 53

R

read 91
renice 58
repquota 55
resize2fs 36
return 91
rmdir 10, 11
rmmod 28
rm 10, 12
route 72
RSA 74
rsync 80-81

S

scp 75
shift 89
sort 95
source 20, 25
sshd 73
ssh 73-76
sudo 50, 51
su 50, 51
/sys 29
sysctl 31

T

tail 20
tar 40, 88, 89
tcptraceroute 72
tee 19
test 18
then 94
top 57
touch 11
traceroute 72
tune2fs 36
  tune2fs.reiserfs 36

U

umask 11, 26, 49
umount 43
unalias 26
uname 6
unique 95
unison 80
updatedb 15
uptime 57
useradd 45, 46
userdel 46
usermod 46
usrquota 53

V

visudo 51, 52
VPN 75

W

wc 19
while 89, 91, 92, 95

X

xfs-dump 37
xfs_check 37
xfs_freeze 37
xfs_growfs 37
xfs_repair 37

Y

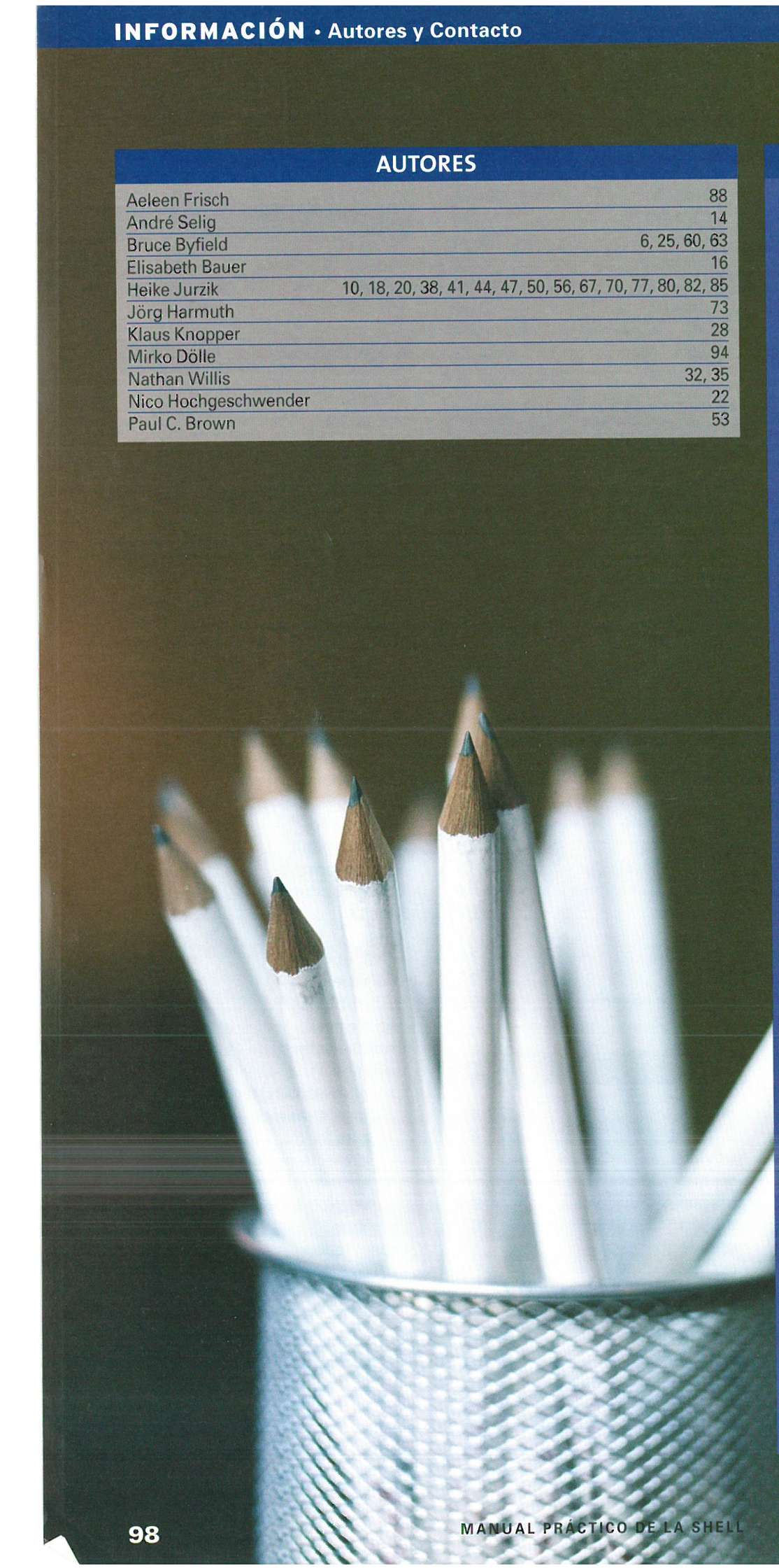
yum 62-66
  check-update 64
  clean 65
    all 65
    dbcache 65
    expire-cache 65
    headers 65
    meta-data 65
    packages 65
  info 65
  groupinfo 65
  groupinstall 64
  groupremove 64
  groupupdate 64
  install 64
  list 65
    availabale 65
    recent 65
    updates 65
  localinstall 64
  make-cache 65
  plugins 66
    allow-downgrades 66
    fastest-mirror 66
    presto 66
    protects-packages 66
    security 66
    version-lock 66
  remove 64
  resolvedep 64
  provides 65
  update 64

Z

zcat 20
zless 20
zmore 20

AUTORES

Aeleen Frisch	88
André Selig	14
Bruce Byfield	6, 25, 60, 63
Elisabeth Bauer	16
Heike Jurzik	10, 18, 20, 38, 41, 44, 47, 50, 56, 67, 70, 77, 80, 82, 85
Jörg Harmuth	73
Klaus Knopper	28
Mirko Dölle	94
Nathan Willis	32, 35
Nico Hochgeschwender	22
Paul C. Brown	53



CONTACTO

Director
Paul C. Brown

Coolaboradores
Paul C. Brown, Klaus Knopper, Heike Jurzik

Traductores
Paqui Martín Vergara, Lucas González, Víctor Tienda, Alberto Planas

Maquetación
Miguel Gómez Molina

Diseño de Portada
Pinball info@pinball-werbeagentur.de

Publicidad
www.linux-magazine.es/pub/

Para España
Marketing y Comunicaciones
anuncios@linux-magazine.es
Tel.: (+ 34) 952 020 242
Fax.: (+ 34) 951 235 905

Para el Resto del Mundo
Hubert Wiest
ads@linux-magazine.com
Tel.: (+49) 8999 34 11 23
Fax.: (+49) 8999 34 11 99

Director Editorial
Paul C. Brown

Jefe de Producción
Miguel Gómez Molina

Subscripciones:
www.linux-magazine.es/magazine/subs
Precios Subscripción
España: 54,90 €
Europa: 64,90 €
Resto del Mundo - Euros: 84,90 €
subs@linux-magazine.es
Tel.: (+34) 952 020 242
Fax.: (+34) 951 235 905

Linux Magazine
Linux New Media Spain, S.L.
Edfco. Hevimar, Planta 2, Ofic. 16
C/Graham Bell nº 6
29590 - Málaga
ESPAÑA
info@linux-magazine.es
Tel.: (+34) 952 020 242
(+34) 951 235 904
Fax.: (+34) 951 235 905

www.linux-magazine.es - España
www.linux-magazine.com - Mundo
www.linux-magazine.co.uk - Reino Unido
www.linux-magazine.com.br - Brasil
www.linux-magazine.pl - Polonia

Si bien se toman todas las medidas posibles para garantizar la precisión del contenido de los artículos publicados en Linux Magazine, la editorial no se hace responsable de imprecisiones aparecidas en la revista. Asimismo, Linux Magazine no comparte necesariamente las opiniones vertidas por sus colaboradores en sus artículos. El riesgo derivado del uso del DVD y el material que contiene corren por cuenta del lector. El DVD es estudiado escrupulosamente para confirmar que está libre de virus y errores.

Copyright y Marcas Registradas © 2010 Linux New Media Spain, S.L. Linux New Media Spain S.L. prohíbe la reproducción total o parcial de los contenidos de Linux Magazine sin su permiso previo y por escrito. Linux es una Marca Registrada de Linus Torvalds.

Conferencia Internacional de

Software Libre Open Source World Conference

MÁLAGA 2010

¡Te esperamos en el mayor evento europeo dedicado al Software Libre!

27 de octubre

Jornada Networking para profesionales de empresas usuarias y proveedoras de servicios tecnológicos y administraciones públicas.

28 de octubre

Jornada divulgativa dirigida al público general.

Inscríbete en: www.opensourceworldconference.com

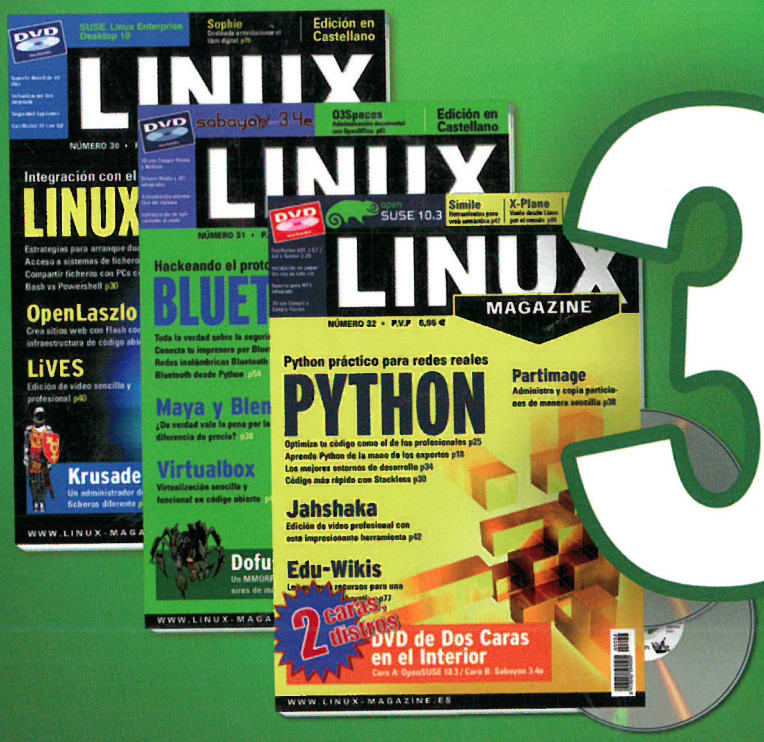
7 y 28 de octubre

MÁLAGA 2010

JUNTA DE EXTREMADURA

JUNTA DE ANDALUCIA

Consigue 3X1



por sólo 6,95€

y además ... con cada número de regalo un fantástico doble DVD 100% Linux

¡ Pídelos ya !

www.linuxmagazine.es/prueba